UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E ELETRÔNICA

Ricardo Peruffo e Silva

MÉTODOS PARA OTIMIZAÇÃO DE UNIDADES DE PROCESSAMENTO ARITMÉTICO RESIDUAL BASEADO EM SELEÇÃO DE MÓDULOS

Florianópolis

2018

Ricardo Peruffo e Silva

MÉTODOS PARA OTIMIZAÇÃO DE UNIDADES DE PROCESSAMENTO ARITMÉTICO RESIDUAL BASEADO EM SELEÇÃO DE MÓDULOS

Trabalho de Conclusão de Curso submetida ao Curso de Graduação em Engenharia Elétrica para a obtenção do Grau de Bacharel em Engenharia Elétrica.
Orientador: Prof. Dr. Héctor Pettenghi Roldán

Florianópolis

2018

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Peruffo e Silva, Ricardo
Métodos para otimização de unidades de processamento aritmético residual baseado em seleção de módulos / Ricardo Peruffo e Silva ; orientador, Héctor Pettenghi Roldán, 2018.
92 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Elétrica, Florianópolis, 2018.

Inclui referências.

1. Engenharia Elétrica. 2. Sistemas numéricos residuais. 3. Unidades aritiméticas. 4. Seleção de módulos. I. Pettenghi Roldán, Héctor. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Elétrica. III. Título.

Ricardo Peruffo e Silva

MÉTODOS PARA OTIMIZAÇÃO DE UNIDADES DE PROCESSAMENTO ARITMÉTICO RESIDUAL BASEADO EM SELEÇÃO DE MÓDULOS

Esta Trabalho de Conclusão de Curso foi julgada aprovada para a obtenção do Título de "Bacharel em Engenharia Elétrica" e aprovada em sua forma final pelo Curso de Graduação em Engenharia Elétrica.

Florianópolis, 25 de junho 2018.

Prof. Dr. Jean Vianei Leite
Coordenador do Curso

Banca Examinadora:

Prof. Dr. Héctor Pettenghi Roldán
Orientador

Prof. Dr. Roberto de Matos
Instituto Federal de Santa Catarina

Engª. Luana Vieira Martinez Bonatto
Universidade Federal de Santa Catarina

RESUMO

Residue Number System é um sistema de representação numérica, que vem sendo pesquisado e utilizado em processos de otimização de operações aritméticas implementadas em *hardware* devido ao sua alto desempenho, principalmente quando a métrica avaliada é o atraso. O alto desempenho está relacionado ao paralelismo inerente às unidades de processamento. Para a realização desse trabalho, foram utilizadas ferramentas computacionais para sintetizar algorítimos implementados em linguagem de descrição de hardware. A partir das sínteses, foram obtidos parâmetros de desempenho referentes às unidades (módulos) implementadas. Sabendo que o sistema residual opera de forma paralela e que são necessários que sejam selecionados agrupamentos (conjuntos) de módulos, os resultados provenientes das sínteses são utilizados para realizar o estudo de técnicas e métodos de seleção e otimização de módulos. Com o foco principal sendo o parâmetro atraso, e sempre preservando a faixa dinâmica do sistema.

Palavras-chave: Sistemas numéricos residuais, unidades aritiméticas, multiplicadores, somadores, seleção de módulos

ABSTRACT

The residue number system is a kind of number representation, that in recent times has been applied in many hardware's arithmetic optimization processes, due to its high performance, specially when it comes to delay. This characteristic is inner to its structure, that is based on making the arithmetic operation most parallel possible. In order to perform this work, computational tools were used to synthesize algorithms based on VHDL. The synthesis results are basically, performance parameters of to the arithmetic units(modules). By knowing the RNS parallel property and that moduli sets are needed to be selected, the synthesis results are used to study techniques and methods to select the optimum set to its specific application. With the focus being on the parameter delay on this paper, knowing also that the dynamic range needs to be preserved.

Keywords: Residue number system, arithmetic unit, multipliers, adders, moduli set selection

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

Sistemas de Numeração Residual (RNS) é um sistema aritméticos livre de *carry* e unidades baseadas em RNS oferecem o potencial de alta velocidade e aplicações em baixa potência para aritmética computacional [1]. As operações aritméticas, como adição, subtração, multiplicação e podem ser realizadas de maneira independente e simultaneamente em vários canais de resíduos de forma mais eficiente do que no sistemas convencionais binários. O RNS tem mostrado várias vantagens no campo de Processamento digital de Sinal, como os filtros digitais [2], sistemas digitais tolerantes a falhas [3], comunicação [4], e criptografia [5].

O mais notório dessas vantagens é a capacidade de paralelizar algoritmos. Esta capacidade é muito significativa na recente tendência de projeto de aceleradores e processadores que se basearam em implementações paralelas massivas, por exemplo, Unidades de Processamento Gráfico. As contribuições relativas ao sistema RNS na literatura podem ser classificados em dois temas principais: i) as operações de binário a RNS e conversão de RNS a binário e ii) algoritmos/operações paralelas que usam a representação RNS [5], [6], [7], [8].

Essa capacidade de paralelização é proveniente da utilização de conjuntos de módulos para que sejam realizadas representação bínarias em aritimética residual. O conjunto RNS mais conhecido é um conjunto definido por três módulos no formato $2^n \pm 1, 2^n$, onde n é o numero de bits dos canais. No caso destes 3 canais, especificamente, a faixa dinâmica será aproximadamente 3n. Com este conjunto de módulos consegue-se arquiteturas RNS muito rápidas e com um número reduzido de recursos quando são implementadas em VLSI. A partir disso, foram apresentados na literatura

conversores binário-RNS e RNS-binário muito eficientes [10].

1.1 MOTIVAÇÃO

Apesar da eficiência inerente aos módulos apresentados acima, eles compõem apenas três módulos. Isso significa que caso seja necessária a implementação de uma faixa dinâmica muito extensa, a aplicação de apenas 3 canais se torna muito limitada e pode não cobrir suficientemente a faixa dinâmica. A partir disso surgem necessidades de implementação de módulos genéricos ($2^n \pm k$), obtendo uma maior paralelização de operações e extenção de faixas dinâmicas. Porém é necessário que seja feito um estudo acerca da implementação de tais módulos, por possuírem uma arquitetura mais complexa que os módulos usuais.

Dessa maneira, torna-se necessário realizar uma análise em relação ao desempenho de diferentes conjuntos de módulos. Toda análise realizada é quantitativa, para que se possa obter resultados concretos em relação às maneiras de selecionar módulos adequados para implementação de projetos otimizados.

1.2 OBJETIVOS

1.2.1 Objetivos Gerais

Análise gráfica/quantitativa de parâmetros de desempenho de unidades aritméticas residuais via implementação em VHDL e síntese ASIC. Visando otimização de blocos aritméticas a partir de uma seleção adequada de módulos.

1.2.2 Objetivos Específicos

- Sintetizar unidades aritiméticas baseadas em arquiteturas bínaria e RNS. Obtendo resultados quantitativos de desempenho em três parâmetros: Atraso, Potência e Área.
- A partir da análise de resultados de sínteses, atraso em especial, exemplificar métodos de escolha de conjunto de módulos que atendam as especificações ótimas.
- Abrir discussão para possíveis trabalhos futuros em seleção de módulos.

2 SISTEMA RNS

O *Residue Number System* (RNS) é um sistema numérico que é definido pela representação de um número inteiro através de um conjunto de números $\{m_1, m_2, m_3, ..., m_n\}$, primos entre si. Ou seja:

$$mdc(m_i, m_j) = 1, \textit{ para todo } i \neq j \tag{2.1}$$

O parâmetro m_i é denominada módulo e o conjunto de módulos pode ser chamada de *moduli*. Para ilustrar, representando um número qualquer X como sendo igual a $(x_1, x_2, x_3, ..., x_n)$, onde:

$$x_i = X \bmod m_i = |X|_{m_i},\ 0 \leq x_i \leq m_i \tag{2.2}$$

Essa representação é única para um número inteiro dentro do intervalo $[0, M-1]$, onde M é chamado de *Dynamic Range*(DR) ou faixa dinâmica do *moduli* $\{m_1, m_2, m_3, ..., m_n\}$. DR pode ser definido como sendo o produto dos módulos do conjunto.

$$M = \prod_{i=1}^{n} m_i \tag{2.3}$$

A seguir é mostrado um exemplo de conversão decimal/residual. Assumindo aleatóriamente $X = 43$ e deseja-se convertê-lo para RNS utilizando, por exemplo, o conjunto $\{3, 7, 10\}$. A primeira verificação a ser feita é se módulos são coprimos:

$mdc(3, 7) = 1$
$mdc(3, 10) = 1$
$mdc(7, 10) = 1$

A partir dessa validação, calcula-se a faixa dinâmica referente ao *moduli*: $M = 3 * 7 * 10 = 210$. Observando assim que é possível representar X dentro da faixa dinâmica, assim realizamos a conversão de X para RNS.

$43 \bmod 3 = 1$
$43 \bmod 7 = 1$
$43 \bmod 10 = 3$

Com os resultados das operações de módulo prontas, o valor 43_{10} é representado por $(1|1|3)_{RNS(3,7,10)}$. Dentro da faixa dinâmica apresentada, pode-se então concluir que essa representação é única dentro do intervalo $[0, 209]$.

A partir das conversões apresentadas. Pode-se perceber que, por ter menor número de bits, operações como soma, subtração e multiplicação devem ser mais rápidas a serem realizadas. Isso ocorre devido ao paralelismo das operações sem que haja propagação de carry entre os digitos residuais.

Para simplificar a implemetação de unidades aritméticas, é sugerido que de módulos sejam representados em potência de dois (binário). Por exemplo o conjunto $\{2^n, 2^n - 1, 2^n + 1\}$, onde as operações estão paralelizadas e cada módulo contribua com $nbits$ para DR, que terá o valor então de $3n$. Ou seja, no caso acima, desejando um $DR = 75bits$ tem-se que $n = 25bits$ e o conjunto de módulos utilizado é $\{2^{25}, 2^{25} - 1, 2^{25} + 1\}$.

Para que sejam realizadas operações aritiméticas utilizando RNS, são necessários fundamentalmente três unidades básicas:

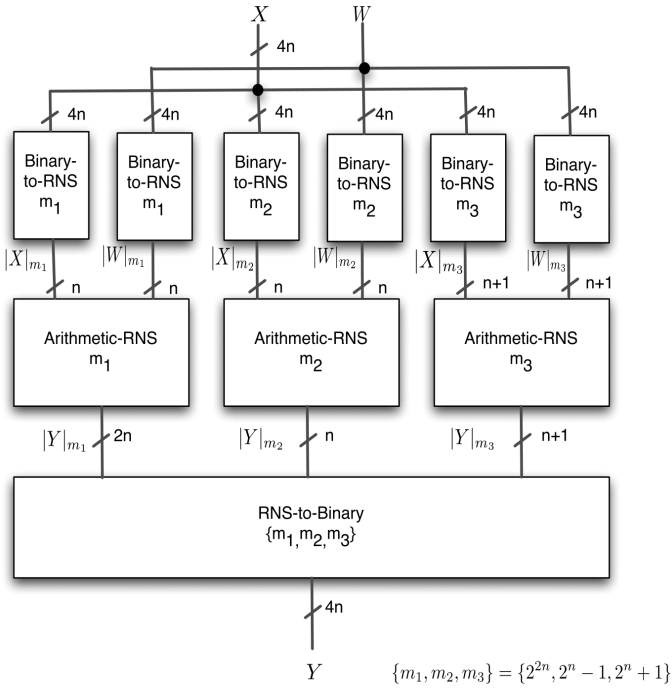
- Conversor direto: Componente que realiza conversão do sistema binário para RNS.

- Unidade aritmética RNS: Componentes que propriamente realizam operações aritiméticas como somas, multiplicações e subtrações em RNS.

- Conversor reverso: Componente que realiza conversão dos resultados das operações em RNS para o resultado final em binário.

A figura 1 representa o diagrama de blocos completo típico RNS, utilizando módulos $\{2^{2n}, 2^n - 1, 2^n + 1\}$[9].

Figura 1 – Representação blocos RNS



Fonte: produzido pelo orientador

2.1 EXEMPLIFICAÇÃO BLOCOS RNS

Nesta seção, será utilizado o conjunto de módulos apresentado na figura 1 para demonstrar a aritmética e implementação de unidades RNS.

2.1.1 Conversão direta

A partir do *moduli* $\{2^{2n}, 2^n - 1, 2^n + 1\}$ um número inteiro $X = \{x_{(4n-1)}, ..., x_1, x_0\}$ pode ser expressado em notação binária como:

$$X = \sum_{i=1}^{4n-1} 2^i x_i = 2^{3n} N_3 + 2^{2n} N_2 + 2^n N_1 + N_0 \qquad (2.4)$$

onde:
$N_3 = \{x_{(4n-1)}, ..., x_{(3n+1)}, x_{3n}\}$
$N_2 = \{x_{(3n-1)}, ..., x_{(2n+1)}, x_{2n}\}$
$N_1 = \{x_{(2n-1)}, .., x_{(n+1)}, x_n\}$
$N_0 = \{x_{(n-1)}, ..., x_1, x_0\}$
Usando notação binária e conjunto de módulos:

$$\{m_1, m_2, m_3\} = \{2^{2n}, 2^n - 1, 2^n + 1\} \qquad (2.5)$$

Tem-se que a faixa dinâmica de X é igual a $[0, M-1]$, onde $M = m_1 m_2 m_3$. Então três conversores são necessários para obter-se a representação RNS, um para cada elemento de base.

- **Canal $m_1 = 2^{2n}$:** O canal mais simples é o conversor usando o modulo m_1. O valor $|X|_{m_1}$ é resultado do resto da divisão de X por 2^{2n}, que pode ser obtida por meio do truncamento do valor de X, uma vez que:

$$|X|_{m_1} = \overbrace{|2^{3n}|_{m_1}}^{=0} N_3 + \overbrace{|2^{2n}|_{m_1}}^{=0} N_2 + 2^n N_1 + N_0 = \{x_{(2n-1)}, ..., x_1, x_0\} \quad (2.6)$$

O canal acima pode ser vizualizado a partir da figura 2a, eliminando o *End-Around-Carry*(EAC), por não possuir *carry*.

- **Canal $m_2 = 2^n - 1$:** Sabendo que $|2^n|_{2^n-1} = 1$, pode-se expressar a equação 2.4 como:

$$|X|_{m_2} = |N_3 + N_2 + N_1 + N_0|_{2^n-1} = |N_3 + |N_2 + N_1 + N_0|_{2^n-1}|_{2^n-1} \quad (2.7)$$

- **Canal $m_3 = 2^n + 1$:** Sabendo que $|2^n|_{2^n+1} = -1$, pode-se expressar a equação. 2.4 como:

$$|X|_{m_3} = |N_3 - N_2 + N_1 - N_0|_{2^n+1} = |-N_3 + |N_2 - N_1 + N_0|_{2^n+1}|_{2^n+1} \quad (2.8)$$

Para a implementação do canal $m_2 = \{2^n - 1\}$ é usada a equação 2.7. O diagrama de blocos para este canal é mostrado na figura 2a.

Cada *Carry-Save Adder* (CSA) soma três termos fornecendo os arrays $C = c_n, ..., c_2, c_1$ e Soma $S = s_{n-1}, ..., s_1, s_0$. Assumindo $|2^n c_n|_{2^n-1} = c_n$, pode-se recolocar o bit c_n na posição 2^0 (i.e: EAC). Finalmente a soma na última etapa é feita por um *Carry-Propagate Adder* (CPA) que também utiliza um EAC.

Para a implementação do canal $m_3 = \{2^n + 1\}$ a partir da equação 2.8. O diagrama de blocos para este canal é mostrado na figura 2a.

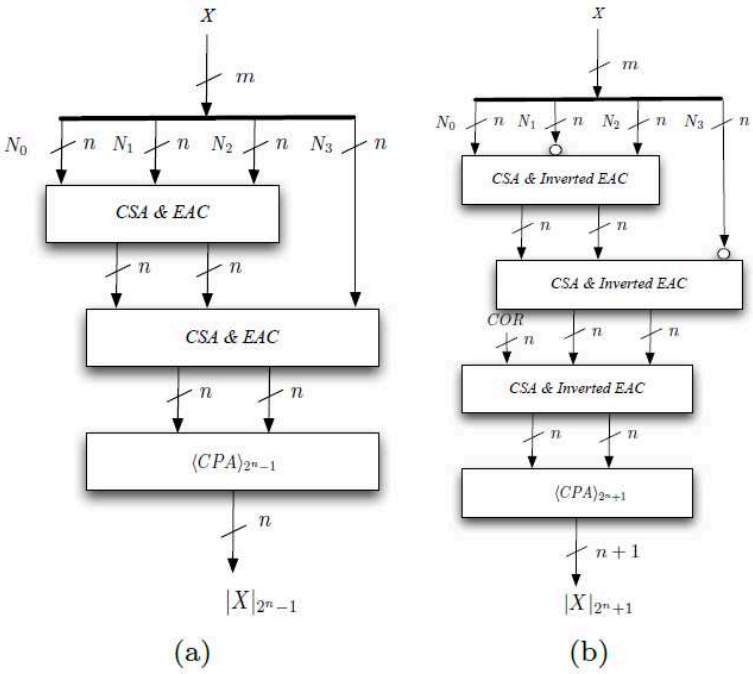
Cada CSA soma novamente três termos fornecendo os barramentos de *carry* $C = c_n, ..., c_2, c_1$ e Soma $S = s_{n-1}, ..., s_1, s_0$. Tendo em consideração que $|2^n c_n|_{2^n+1} = ||2^n|_{2^n+1} c_n|_{2^n+1} = |-c_n|_{2^n+1} = |COR_{level-j} + \bar{c_n}|_{2^n+1}$, pode-se recolocar o bit c_n na posição 2^0 de forma complementada (i.e: operação

de *Inverted* EAC ou IEAC) adicionando um fator corretor $COR_{level-j}$, onde j define qual nível de CSA com inverted EAC está associado ao fator corretor. $COR_{level-j}$ pode ser calculado a partir da seguinte equação $|COR_{level-j} + \bar{c_n}|_{2^n+1} = 0$ quando $c_n = 0$ (i.e. $\bar{c_n} = 1$). Desta forma $COR_{level-j} = 2^n$ por nível CSA-IEAC.

Os termos negativos $-N_0$ e $-N_2$ seguem a mesma regra $|-N_i|_{2^n+1} = |COR_{Ni} + \bar{N_i}|_{2^n+1}$, onde i define qual *array* de N_i está associado ao fator corretor. O fator corretor COR_{Ni} pode ser calculado a partir da seguinte equação $|COR_{Ni} + \bar{N_i}|_{2^n+1} = 0$ quando $N_i = 0$ (i.e. $\bar{N_i} = 2^n - 1$). Desta forma $COR_{Ni} = 2$ para cada $-N_i$.

O fator corretor (COR) final consiste na soma modular de todas as correções parciais $COR = |\sum_{j=1,2,3,4} COR_{level-j} + \sum_{i=0,2} COR_{Ni}|_{m_3}$.

Figura 2 – Conversão direta $2^n \pm 1$



Fonte: produzido pelo orientador

2.1.2 Unidades Aritméticas

Nesta seção serão apresentadas estruturas e aritmética de canais residuais para operações de soma e multiplicação, utilizando os módulos 2^n, $2^n - 1$ e $2^n + 1$[10].

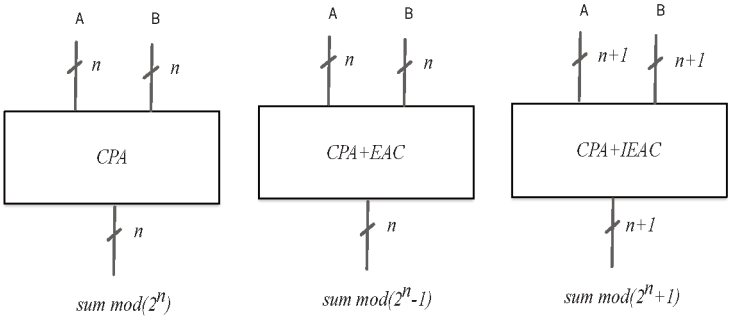
2.1.2.1 Somadores

- **Canal $m_1 = 2^n$:** O valor da soma $S_1 = |Y|_{m_1} = |R_1 + Q_1|_{m_1}$, onde $R_1 = |X|_{m_1} = \{r_{1,(2n-1)}, \ldots, r_{1,0}\}$ e $Q_1 = |W|_{m_1} = \{q_{1,(2n-1)}, \ldots, q_{1,0}\}$ pode ser obtido por meio de um CSA sem EAC, uma vez que $|2^n|_{2^n} = 0$.

- **Canal $m_2 = 2^n - 1$:** Neste caso o valor da soma $S_2 = |Y|_{m_2} = |R_2 + Q_2|_{m_2}$, onde $R_2 = |X|_{m_2} = \{r_{2,(n-1)}, \ldots, r_{2,0}\}$ e $Q_2 = |W|_{m_2} = \{q_{2,(n-1)}, \ldots, q_{2,0}\}$ pode ser obtido por meio de um CSA adicionado a um EAC, uma vez que $|2^n|_{2^n-1} = 1$.

- **Canal $m_3 = 2^n + 1$:** Neste caso o valor da soma $S_3 = |R_3 + Q_3|_{m_3}$, onde $R_3 = |X|_{m_3} = \{r_{3,(n)}, \ldots, r_{3,0}\}$ e $Q_3 = |W|_{m_3} = \{q_{3,(n)}, \ldots, q_{3,0}\}$ é obtido através de um CSA adicionado a um IEAC, uma vez que $|2^n|_{2^n+1} = -1$

Figura 3 – Estrutura somadores 2^n $2^n \pm 1$

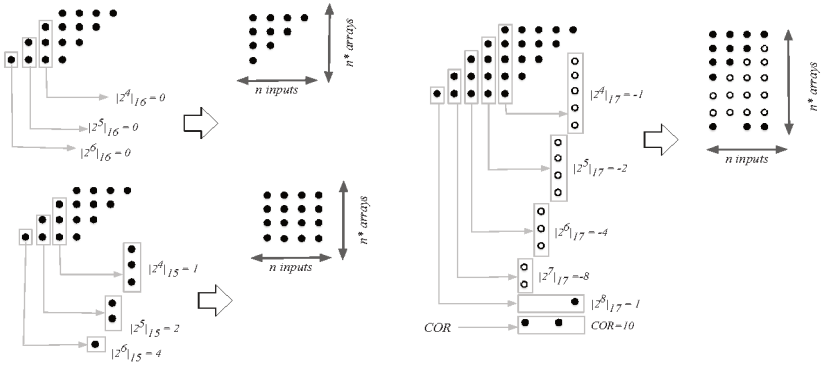


Fonte: produzido pelo orientador

2.1.2.2 Multiplicadores

Seguindo o exemplo proposto, são necessários três multiplicadores para obter-se a representação do RNS. A figura 4 exemplifica como ocorrem as multiplicações para os devidos módulos, usando a notação de pontos. Os pontos pretos correspondem aos produtos parciais da operação de multiplicação, enquanto os brancos correspondem aos complementos deles. Para este exemplo, foi adotado o valor de $n = 4bits$.

Figura 4 – Multiplicações RNS 2^n $2^n - 1, 2^n + 1$



Fonte: produzido pelo orientador

- **Canal $m_1 = 2^n$:** Neste caso o valor da multiplicação $S_1 = |Y|_{m_1} = |R_1 \times Q_1|_{m_1}$, onde $R_1 = |X|_{m_1} = \{r_{1,(2n-1)}, \ldots, r_{1,0}\}$ e $Q_1 = |W|_{m_1} = \{q_{1,(2n-1)}, \ldots, q_{1,0}\}$ pode ser obtido através do truncamento dos valores de carry da multiplicação, pois $|2^n|_{2^n} = 0$ como mostrado na Figura 4.

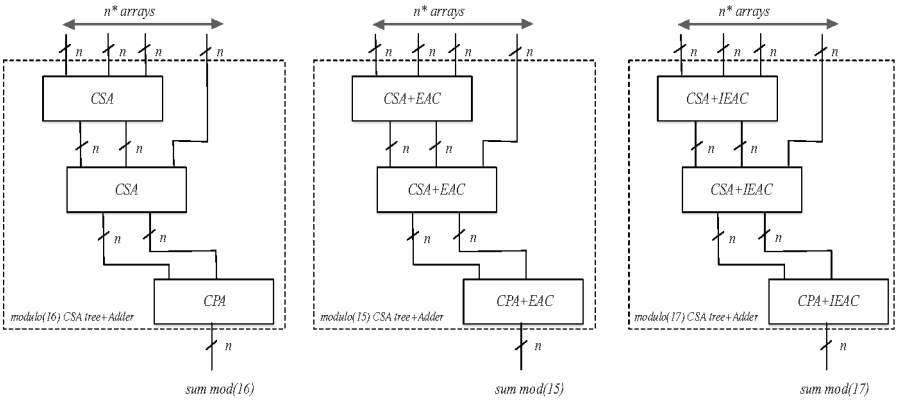
- **Canal $m_2 = 2^n - 1$:** Neste caso o valor da multiplicação $S_2 = |Y|_{m_2} = |R_2 + Q_2|_{m_2}$, onde $R_2 = |X|_{m_2} =$

$\{r_{2,(n-1)}, \ldots, r_{2,0}\}$ e $Q_2 = |W|_{m_2} = \{q_{2,(n-1)}, \ldots, q_{2,0}$ é obtido aplicando EAC em todos os níveis de soma realizado, redistribuindo os valores de carry e os somando em seu devido nível pois $|2^{n+i}|_{2^n-1} = 2^i$ como mostrado na Figura 4.

- **Canal $m_3 = 2^n + 1$:** Neste caso o valor da multiplicação $S_3 = |R_3 + Q_3|_{m_3}$, onde $R_3 = |X|_{m_3} = \{r_{3,(n)}, \ldots, r_{3,0}\}$ e $Q_3 = |W|_{m_3} = \{q_{3,(n)}, \ldots, q_{3,0}$ é obtido aplicando dessa vez *Inverted* EAC nos níveis de soma, uma vez que $|2^{n+i}|_{2^n+1} = -i$. Além disso, é preciso incluir um fator corretor como mostrado na Figura 4.

A figura 5 traz os diagramas de blocos de implementações de tais módulos. O diagrama do módulo $2^n + 1$ foi simplificado para um número reduzido de quatro *arrays*. Porém na realidade seriam necessários sete *arrays* como mostrado na figura 4.

Figura 5 – Diagramas de blocos multiplicação RNS 2^n, $2^n \pm 1$



Fonte: produzido pelo orientador

2.1.3 Conversão Reversa

A conversão de RNS a binário pode ser obtida utilizando o Teorema Chinês do Resto (CRT).

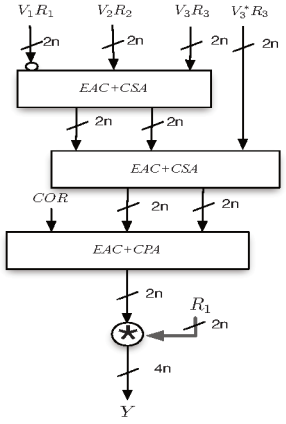
$$X = \left| \sum_{i=1}^{3} \hat{m}_i \left| \hat{m}_i^{-1} \right|_{m_i} R_i \right|_M , \tag{2.9}$$

onde $\hat{m}_i = \frac{M}{m_i}$, $\left|\hat{m}_i^{-1}\right|_{m_i}$ é a multiplicativa inversa de $\hat{m}_i$ e R_i representa as entradas residuais. Tais multiplicativas inversas são números inteiros que satisfazem a condição

$$\left| \left| \hat{m}_i^{-1} \right|_{m_i} \hat{m}_i \right|_{m_i} = 1 \tag{2.10}$$

Para o *moduli* $\{2^{2n}, 2^n - 1, 2^n + 1\}$, obtemos o bloco apontado na figura 1. Os valores das constantes V_i e COR podem ser encontradas em [9].

Figura 6 – Conversor reverso $\{2^{2n}\ 2^n - 1, 2^n + 1\}$



2.2 UNIDADEDES ARITMÉTICAS $2^N \pm K$

A seção 2.1 demonstra o funcionamento de um operador aritmético RNS completo. Para exemplificação, dentro de um $DR = 4n$, foram utilizados módulos do tipo 2^{2n}, $2^n - 1$ e $2^n + 1$. Com finalidade de oferecer uma unidade aritmética com mais paralelismo, onde o DR seja distribuido entre mais módulos, é necessária a utilização de diversos tipos de módulos.

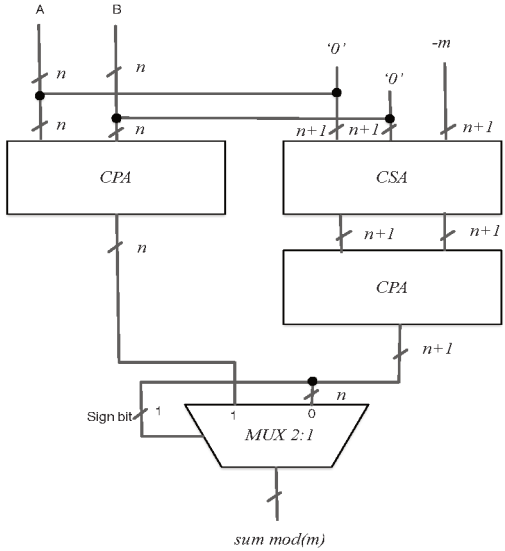
Trabalhando em potências de dois, são sugeridos então os módulos no formato $2^n - k$ e $2^n + k$.

Inserindos esses módulos no conjunto de módulos usado na seção anterior. O novo conjunto de módulos é $\{2^{2n}, 2^n - 1, 2^n + 1, 2^n - 3, 2^n + 3\}$, onde $k = 3$.

Apesar do sistema como um todo operar em maior paralelismo. Tem-se que um maior número de módulos, pode causar o aumento da complexidade na arquitetura do conversor reverso, assim aumentando seu atraso.

É conhecido também que o caminho crítico certamente passará pelos novos módulos, devido à sua arquitetura. Para justificar essa afirmação, primeiramente serão exibidos os blocos somadores $2^n \pm k$, com uma breve explicação do funcionamento. Em seguida será mostrada como realizar multiplicações $2^n \pm k$ e sua arquitetura relacionada.

Figura 7 – Somador $2^n \pm k$

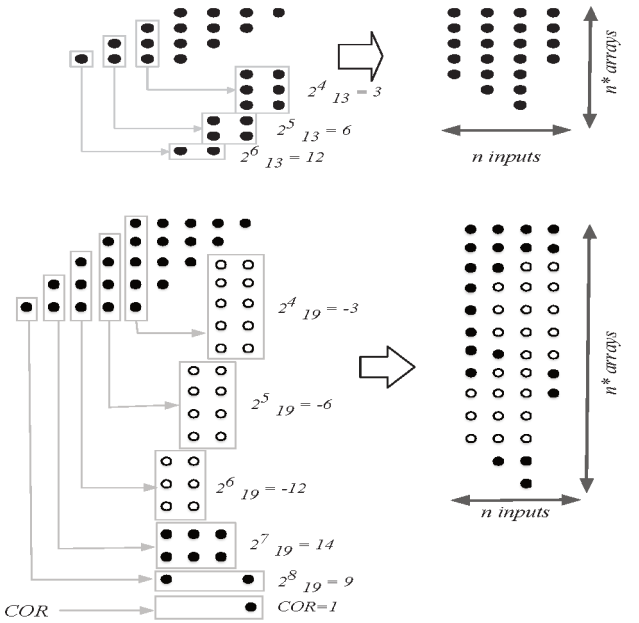


Fonte: produzido pelo orientador

A operação realizada na figura 7 é basicamente $A+B-m$ se $A+B \geq m$, caso contrário o resultado é igual a $A+B$ [14].

A multiplicação $2^n \pm k$ será exemplificada adotando $k = 3$ e $n = 4$, portanto, na figura 8:

Figura 8 – Multiplicação $2^4 + 3$

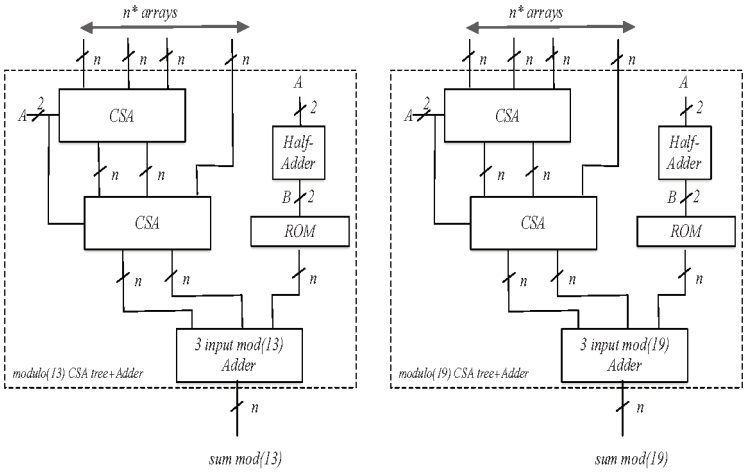


Fonte: produzido pelo orientador

Na figura 8 é mostrada a pré-computação dos produtos parciais para módulos 13 $(2^4 - 3)$ e 19 $(2^4 + 3)$. Uma vez mostrada a matriz de $n^* \cdot n$ bits pode-se então realizar a soma modular n^* vetores de n bits, como apresentado na figura 9.

Vale ressaltar que na figura 9, o tamanho do *array* foi reduzido para $n^* = 4$ para facilitar a representação da figura.

Figura 9 – Diagrama de blocos multiplicador $2^n \pm k$



Fonte: produzido pelo orientador

Somando os n^* vetores e as respectivas contribuições dos *carrys* somados e endereçados em uma *Read-Only Memory* (ROM). A soma final $mod(n)$ de três vetores é feita ao fim para obter o vetor de multiplicação modular [11][12][13].

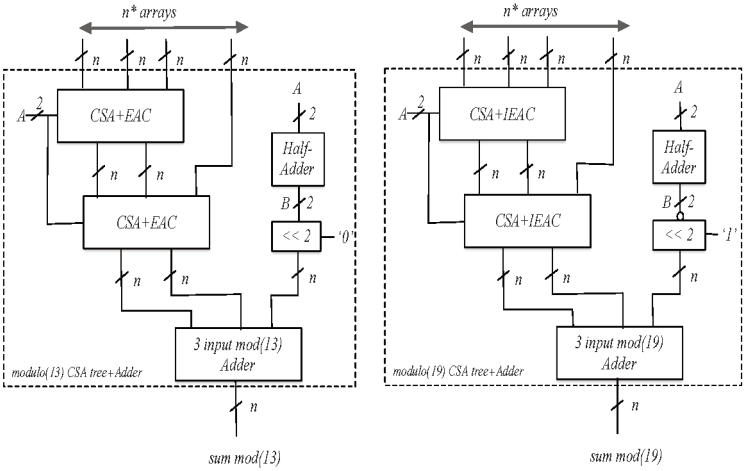
É fácil comparar que as implementações $2^n, 2^n \pm 1$ das figuras 3 e 5 são mais eficientes que as arquiteturas $2^n \pm k$ apresentadas nas figuras 7 e 9, devido ao número de componentes utilizados.

Sabendo das limitações inerentes às arquiteturas acima. O grupo de pesquisa da UFSC em RNS, liderado pelo Prof. Dr. Héctor Roldán, recentemente propôs um caso específico de módulos $2^n \pm k$. É proposta então a decomposição $k = 2^\alpha \pm 1$, ou seja, os novos módulos no formato $2^n \pm 2^\alpha \pm 1$. α sendo um número inteiro

Por possuirem arquitetura mais simples, assim reduzind o caminho crítico quando comparados aos módulos $2^n \pm$

k genéricos. As arquiteturas referentes aos novos módulos são mostradas na figura 10.

Figura 10 – Diagrama de blocos mutliplicador $2^n \pm 2^\alpha \pm 1$



Fonte: produzido pelo orientador

Como é demonstrado acima, a memória ROM é eliminada. Visto que a contribuição de ∓ 1 é realizada nos EAC e $IEAC$, respectivamente. Além disso as contribuições $\mp 2^\alpha$ são realizadas em um simples deslocamento ou complemento dele, respectivamente.

2.3 SELEÇÃO DE MÓDULOS

Selecionar módulos adequados possui um papel importante no projeto de sistemas RNS, visto que a implementação de novos módulos causa mudanças na arquitetura da unidade RNS de maneira geral. Tais mudanças podem causar variações de:

- Caminho crítico
- Faixa dinâmica
- Área combinacional
- Complexidade do conversor reverso
- Potência dinâmica

Por isso é importante que sejam bem específicadas as condições de projeto, para que assim seja possível selecionar os melhores módulos que atendam os parâmetros desejados.

Sabendo disso, foram implementados em $VHDL$ unidades multiplicadoras por constante RNS de módulos 2^n, $2^n \pm 1$, $2^n \pm k$ e $2^n \pm 2^\alpha \pm 1$. Esses módulos foram sintetizados em ferramenta computacional e os próximos dois capítulos tratam de mostrar resultados referentes às sinteses e trazem métodos de como realizar as devidas seleções de módulos.

3 IMPLEMENTAÇÃO E RESULTADOS

3.1 INTRODUÇÃO

Este presente capítulo, que tem como objetivos:

- Apresentação de resultados, numéricos e gráficos, referentes à implementação das unidades lógicas sintetizadas.
- Apresentação da ferramentas utilizadas na implementação das unidades lógicas.
- Exemplificação de problemas referentes à implementação de arquiteturas binárias convencionais, comparando aos resultados obtidos por implementações RNS.
- Introdução ao próximo capítulo referente a seleção de módulos através de uma análise quantitativa.

3.2 FERRAMENTA UTILIZADA

Os resultados foram obtidos visando aplicação em ASIC's, utilizando a biblioteca TCBN65GLPLUS, versão 200 para TSMC 65nm. Os arquivos foram sintetizados utilizando o Cadence RTL Compiler tools (versão v11.20-s012_1).

3.3 PROBLEMATIZAÇÃO E IMPLEMENTAÇÃO EM BINÁRIO VERSUS RNS

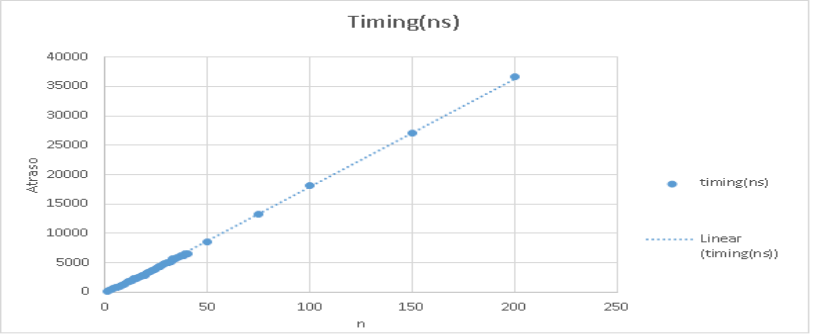
Os apêndices A1 até A7 apresentam resultados referentes às sínteses de unidades multiplicadoras por variável de

arquitetura binária e RNS. Em binário para $n \geq 40bits$ apenas alguns valores pontuais de n foram sintetizados. Dessa maneira foi possível traçar curvas de regressão, funções que estimam os valores das caracterísiticar físicas em função de n. Essas funções são utilizadas para valores de n de até 200 bits.

Já em RNS, para as unidades 2^n e $2^n \pm 1$ foram sintetizadss para $5 \leq n \leq 65$ exclusivamente ímpares. E por apresentarem arquiteturas mais complexas, os demais módulos foram síntetizados para $5 \leq n \leq 35$, também ímpares.

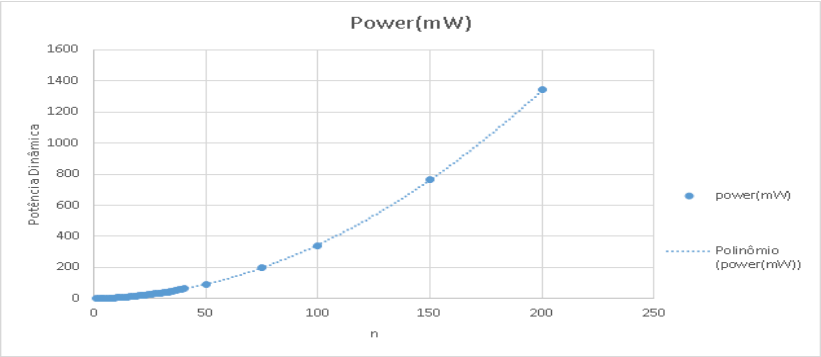
O mesmo procedimento de curvas de regressão foi adotado, para os módulos RNS. As equações utilizadas encontram-se no apêndice A9. Abaixo encontram-se os gráficos das arquiteturas binária e RNS (com regressões aplicadas)

Figura 11 – Atraso por número de bits(binário)



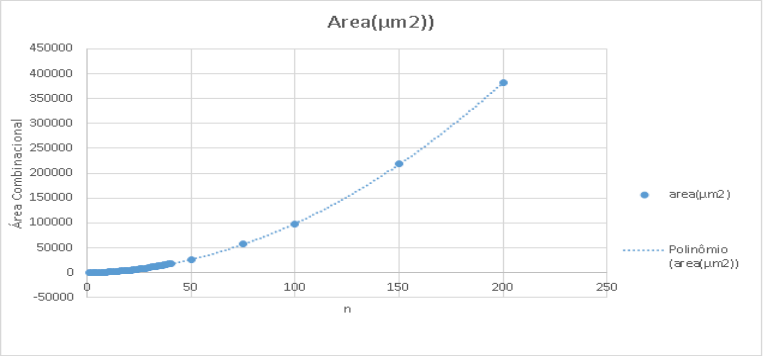
Fonte: produzido pelo autor

Figura 12 – Potência por número de bits(binário)



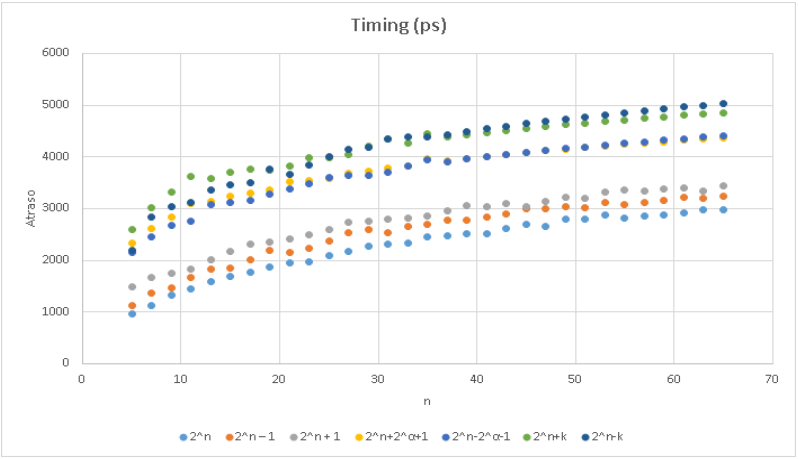
Fonte: produzido pelo autor

Figura 13 – Área por número de bits(binário)



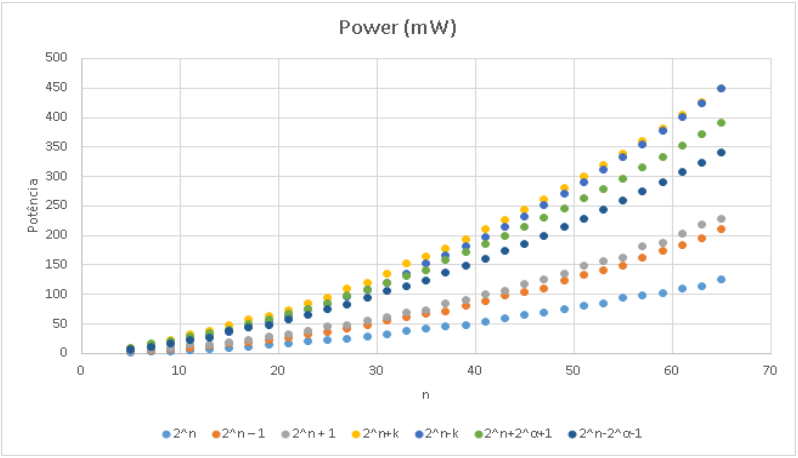
Fonte: produzido pelo autor

Figura 14 – Atraso por número de bits(RNS)



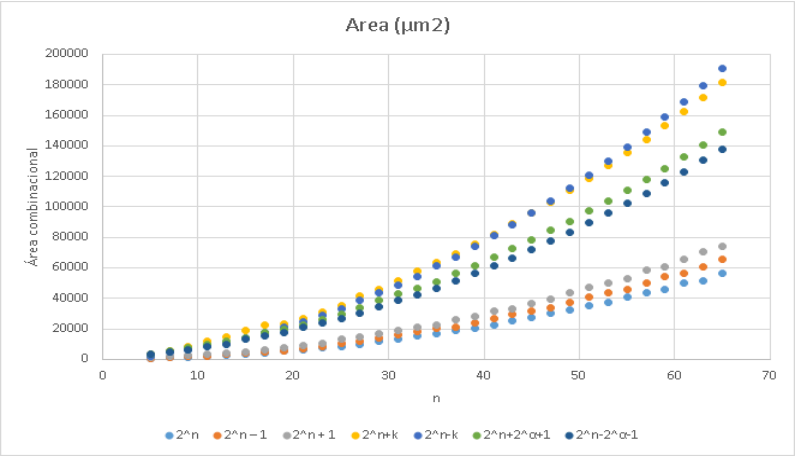
Fonte: produzido pelo autor

Figura 15 – Potênca por número de bits(binário)



Fonte : produzido pelo autor

Figura 16 – Área por número de bits(binário)



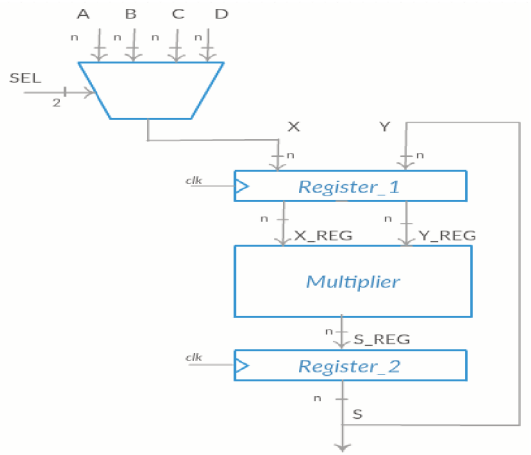
Fonte: produzido pelo autor

Abaixo estão exemplificados dois casos onde, baseando-se nas tabelas e equações apresentadas, serão realizados cálculos de desempenho referentes a implementação de tais componentes utilizando arquiteturas binário e RNS.

3.3.1 Caso 1

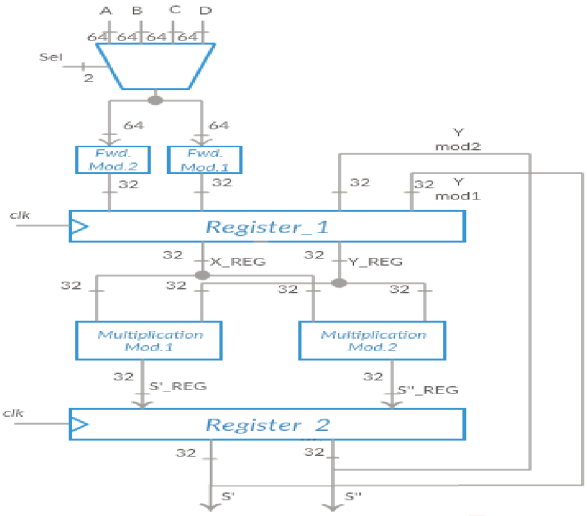
Deseja-se que seja implementado uma unidade aritmética que possua entrada e saída registrada e que tenha a função de realizar a multiplicação de quatro valores de entrada. Na figura 17 é mostrado a implementação realizada em binário e na figura 18 a proposta em RNS adotando um conjunto de dois módulos. Para obter-se o valor de uma multiplicação em binário, na figura 18, as saídas S' e S'' devem ser submetidas por um conversor reverso.

Figura 17 – Diagrama de blocos(binário) caso 1



Fonte: Produzido pelo autor

Figura 18 – Diagrama de blocos(RNS)/Realimentado



Fonte: Produzido pelo autor

A tabela 16 apresentada no apêndice A.10 demonstra o comportamento de entradas, saídas e sinais internos da unidade multiplicadora nos instantes mais próximos possíveis de bordas de subida. A partir dessa análise é possível calcular o clock necessário para realizar as operações e também o tempo total após todas iterações proposta. Essa análise é necessária devido aos registradores posicionados nas entradas e saídas das unidades aritiméticas.

A partir de tal análise, é possível concluir que:

$$T_{clock} \geq D(n) \tag{3.1}$$

Adotando vetores de 64 bits, e aplicando esse valor às equações de regressão apresentadas no apêndice A9. Escolhendo o conjunto de dois módulos balanceados mais otimizado para $DR = 64bits : \{2^{32}, 2^{32} - 1\}$. Calcula-se que:

Binário:
$Atraso_{bin} = D(64) \approx 11,33ns$
$Potncia_{bin} = P(64) \approx 147,92mW$
$rea_{bin} = A(64) \approx 42297,40\mu m^2$

RNS:
$Atraso_{2^{32}} = 2,33ns$
$Atraso_{2^{32}-1} = 2,60ns$
$Potncia_{2^{32}} = 39,20mW$
$Potncia_{2^{32}-1} = 59,20mW$
$rea_{2^{32}} = 14230,40\mu m^2$
$rea_{2^{32}-1} = 17000,00\mu m^2$

Logo, o período de relógio para binário é igual a $11,3ns$ e para RNS $2,60ns$

Para que sejam efetuadas as quatro multiplicações serão necessários 7 ciclos de clock (apêndice A10). Então

conclui-se que:

Tabela 1 – Comparativo implementação 1

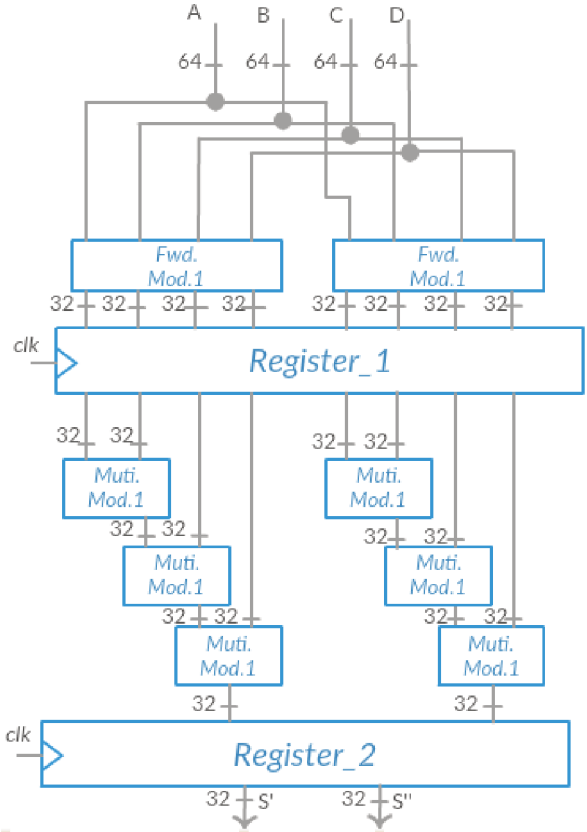
Parâmetro	Binário	RNS	**Ganho (%)**
Atraso	79,10 ns	18,30 ns	**76,87**
Potência	591,68 mW	393,60 mW	**33,47**
Área	42297,40 μm^2	31230,40 μm^2	**26,16**

Fonte: produzido pelo autor

Vale ressaltar que apesar de que aumentando o número de operações, o valor relativo entre o atraso total binário e RNS se mantém o mesmo. Em valores absolutos quanto mais operações feitas, maior será a diferença entre os atrasos das duas arquiteturas.

A seguinte proposta (figura 19) também pode ser aplicada ao caso apresentado, nessa idéia é os quatro vetores serão multiplicados em apenas um ciclo de relógio. Nesse caso é utilizado o mesmo conjunto de módulos do exemplo anterior.

Figura 19 – Diagrama de blocos(RNS)/Cascata



Fonte: produzido pelo autor

Realizando os cálculos de atraso, potência, e área para esse diagrama e comparando com os valores obtidos nas duas outras configurações, tem-se que:

$$T_{clock} = Atraso_{Total} = 3 \cdot 2,60 = 7,80ns \tag{3.2}$$

$$Potncia_{Total} = 3 \cdot 39,2 + 3 \cdot 59,2 = 259,2mW \tag{3.3}$$

$$rea_{Total} = 3 \cdot 14230,40 + 3 \cdot 17000,00 = 93691,60\mu m^2 \tag{3.4}$$

Comparando resultados atuais com os apresentados em binário:

Tabela 2 – Comparativo implementação 2

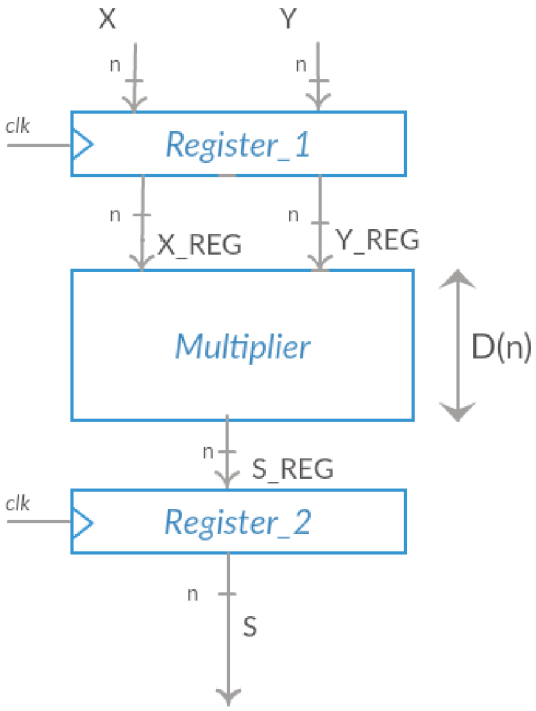
Parâmetro	Binário	RNS2	**Ganho** (%)
Atraso	79,10 ns	7,80 ns	**91,41**
Potência	591,68 mW	259,20 mW	**43,80**
Área	42297,40 μm^2	63691,60 μm^2	$-50{,}58$

Fonte: produzido pelo autor

3.3.2 Caso 2

Utilizando uma unidade de multiplicação (figura 20), sem realimentação. Considere que tal unidade multiplicadora está inserida em um contexto de frequência de clock igual a 200 MHz ou período igual à 5 ns.

Figura 20 – Diagrama de blocos(binário) caso 2



Fonte: produzido pelo autor

Considerando $n = 128bits$ para esse caso, utilizando as equações do apêndice A8:

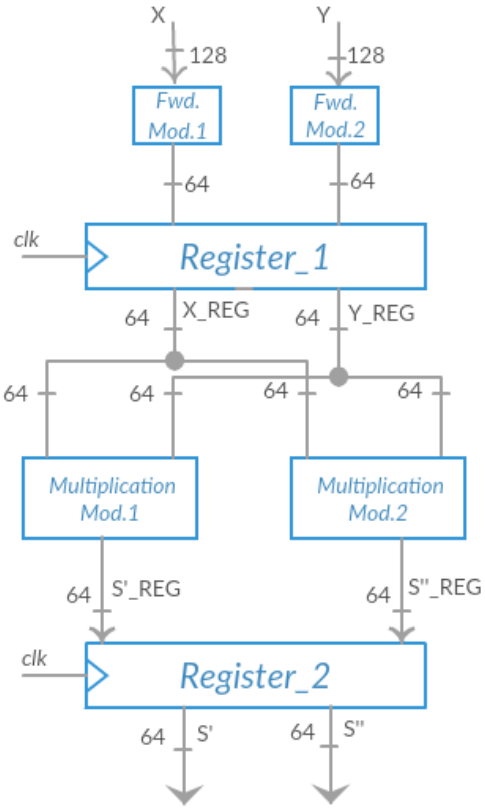
$D(128) \approx 23,01ns$
$P(128) \approx 561mW$
$A(128) \approx 160096,96\mu m^2 s$

Neste caso com *clock* definido e menor que o atraso para realização da multiplicação, a implementação dessa unidade

aritmética não retornaria valores confiáveis caso ocorram alterações nas entradas do registrador entre uma operação e outra.

Descartando a possíbilidade do uso de arquitetura binária, é proposta a seguinte solução utilizando os módulos $\{2^{64}, 2^{64}-1\}$:

Figura 21 – Diagrama de blocos(RNS)/Direto



Fonte: produzido pelo autor

Obtem-se então que:

$Atraso_{2^{64}} = 2,97ns$
$Atraso_{2^{64}-1} = 3,23ns$
$Potncia_{2^{64}} = 125,59mW$
$Potncia_{2^{64}-1} = 211,127mW$
$Potncia_{Total} = 336,32mW$
$rea_{2^{64}} = 56228,75\mu m^2$
$rea_{2^{64}-1} = 65926,43\mu m^2$
$rea_{Total} = 122155,18\mu m^2$

Traçando um comparativo entre as duas aplicações:

Tabela 3 – Comparativo implementação 3

Parâmetro	Binário	RNS	**Ganho (%)**
Atraso	23,01 ns	3,23 ns	**85,96**
Potência	561 mW	336,32 mW	**40,05**
Área	160096, 96 μm^2	65926,43 μm^2	**58,82**

Fonte: produzido pelo autor

O caminho crítico passa pelo módulo $2^{64} - 1$, e esse caminho cabe nas condições iniciais do período de *Clock*. Portanto esse conjunto de módulos é uma solução valida.

3.4 CONSIDERAÇÕES SOBRE SELEÇÃO DE MÓDULOS

Apesar do conjunto de módulos acima ter resolvido a problemática, não existe garantia de que esse seja o módulo mais otimizado em qualquer parâmetro (atraso, potência e área).

A partir desse momento o parâmetro atraso será tomado como base dos métodos e resultados a serem apresentados.

A partir do cálculo do primeiro conjunto de módulos, duas linhas de raciocínio são possíveis para que o circuito possa funcionar na forma mais otimizada possível. A primeira maneira é o aumento do número de bits dos conjuntos de módulos, o que faz com que a faixa dinâmica se estenda.

Na condição do exemplo acima, onde $\Delta \leq 5ns$. Onde Δ é o caminho crítico do *moduli*

Ao observar a figura 14, nota-se que é possível estender n até valores próximos a 100 bits, que a condição acima se manterá. Isso é possível pois esses módulos são coprimos para qualquer valor de n inteiro e positivo.

A segunda proposta de otimização consiste em encontrar o conjunto de módulos que possua o menor caminho crítico (δ).
No exemplo acima foi utilizado o conjunto de módulos $\{2^n; 2^n - 1\}$, com $n = 64$ e $DR = 128$(resultados referentes à figura 21). Adicionando o módulo $2^n + 1$, em busca de aumentar o paralelismo da operação, mantendo o $DR = 128bits$ e distribuindo a faixa dinâmica igualmente entre módulos. O novo conjunto passa a ser: $\{2^{43}; 2^{43} - 1; 2^{43} + 1\}$

Cujos atrasos associados são:
$Atraso_{2^{43}} = 2,61ns$
$Atraso_{2^{43}-1} = 2,90ns$
$Atraso_{2^{43}+1} = 3,10ns$

Obseva-se então uma redução do caminho crítico em aproximadamente 4%.

A partir desse resultado, o próximo capítulo aborda de maneira gráfica/quantitativa a questão de otimização via seleção de módulos com foco na redução do caminho crítico.

Lembrando também que nessa análise não está sendo considerado o atraso do conversor reverso, que nesse caso seu efeito seria o de um *offset* ao caminho crítico. O que pode mudar o resultado final, mas o método de análise é o mesmo.

4 SELEÇÃO DE MÓDULOS

Nesse capítulo são apresentados procedimentos sistemáticos para que seja encontrado o conjunto de módulos que possua o menor caminho crítico, que de fato é a maior contribuição deste trabalho. Nas seguintes seções serão mostradas as seguintes formas de seleção de módulo:

- 4.1.1) Seleção de conjunto de módulos usando $2^n, 2^n \pm 1, 2^n \pm 2^\alpha \pm 1, 2^n \pm k$
- 4.1.2) Seleção de conjunto de módulos usando $2^n, 2^n \pm 1, 2^n \pm 2^\alpha \pm 1$
- 4.1.3) Seleção de conjunto de módulos usando $2^n, 2^n \pm 1$
- 4.1.4) Seleção de conjunto de módulos usando $2^n, 2^n - 1$
- 4.2) Formas de otimização para as diferentes formas de seleção apresentadas.

4.1 SELEÇÃO DE CONJUNTOS DE MÓDULOS

Por se tratar da unidade aritimética mais complexa analisada e para poder traçar gráficos a partir de resultados numéricos consistentes, foram adotadas como exemplo unidades multiplicadoras por variável (sintetizadas) e também determinado $DR = 128bits$.

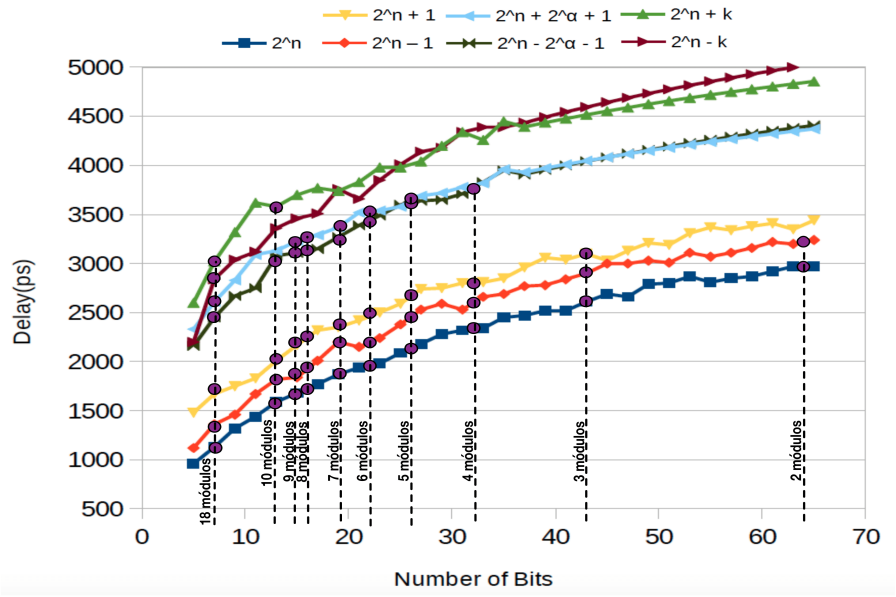
4.1.1 Seleção de conjunto de módulos usando $2^n, 2^n \pm 1, 2^n \pm 2^\alpha \pm 1, 2^n \pm k$

Nessa subseção é realizado um processo iterativo que consiste em adicionar módulos coprimos ao conjunto em ques-

tão, utilizando todas as arquiteturas RNS sintetizadas. O método é iniciado no menor conjunto de módulos possível, em seguida é adicionado um novo módulo coprimo e a faixa dinâmica é redistribuida. Esse processo é repetido até o momento onde não é mais viável adicionar módulos. Dessa maneira é possível avaliar nas devidas condições qual é o conjunto de módulos que apresenta menor caminho crítico, além de extrair outras informações.

O gráfico da figura 22 indica as iterações realizadas nesse processo, variando o conjunto de dois até dezoito módulos.

Figura 22 – Atraso por número de bits(RNS)

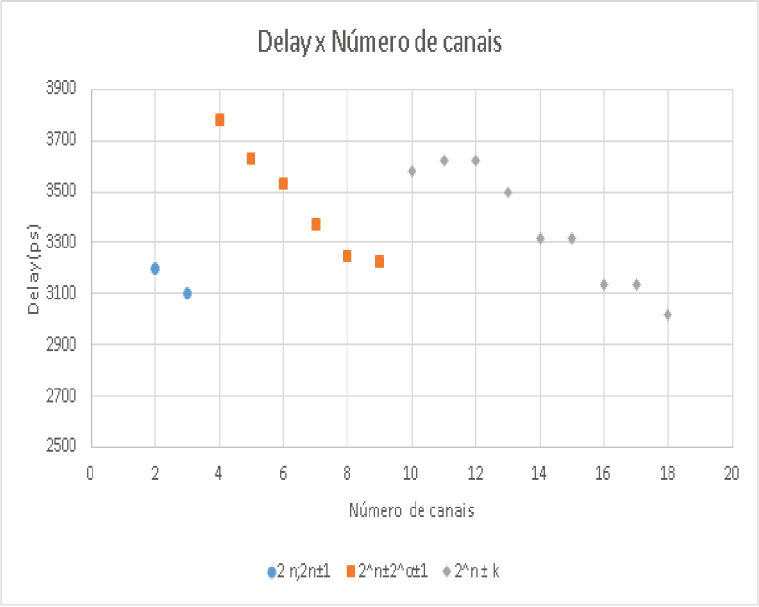


Fonte: produzido pelo autor

Já o gráfico da figura 23 exibe o caminho crítico relativo à quantidade de módulos no conjunto, indicando também

qual é a arquitetura referente ao maior atraso.

Figura 23 – Caminho crítico por número de canais - 1



Fonte: produzido pelo autor

Detalhando ainda mais o processo, a tabela 4 exibe os conjuntos de módulos utilizados nessa subseção, enquanto na tabela 5 encontram-se os valores exatos referentes ao caminho crítico . Foi realizada uma pesquisa dos módulos utilizados até o conjunto de dez módulos, após essa quantidade foi assumido que o caminho crítico seria pelo módulo $2^n + k$ por seus valores serem os mais altos em atraso em $7 \leq n \leq 13$.

Tabela 4 – Conjuntos de módulos 4.1.1

qtd mod	conjunto
2	$2^n; 2^n - 1$
3	$2^n; 2^n - 1; 2^n + 1$
4	$2^n; 2^n - 1; 2^n + 1; 2^n - 3$
5	$2^n; 2^n - 1; 2^n + 1; 2^n - 3; 2^n + 3$
6	$2^n; 2^n - 1; 2^n + 1; 2^n - 3; 2^n + 3; 2^n - 5$
7	$2^n; 2^n - 1; 2^n + 1; 2^n - 3; 2^n + 3; 2^n + 5; 2^n - 7$
8	$2^n; 2^n - 1; 2^n + 1; 2^n - 3; 2^n + 3; 2^n - 5; 2^n + 7; 2^n - 9$
9	$2^n; 2^n - 1; 2^n + 1; 2^n - 3; 2^n + 3; 2^n + 5; 2^n - 7; 2^n - 9; 2^n + 9$
10	$2^n; 2^n - 1; 2^n + 1; 2^n - 3; 2^n + 3; 2^n + 5; 2^n + 11; 2^n - 13; 2^n + 15; 2^n + 17$

Tabela 5 – Caminho crítico 4.1.1

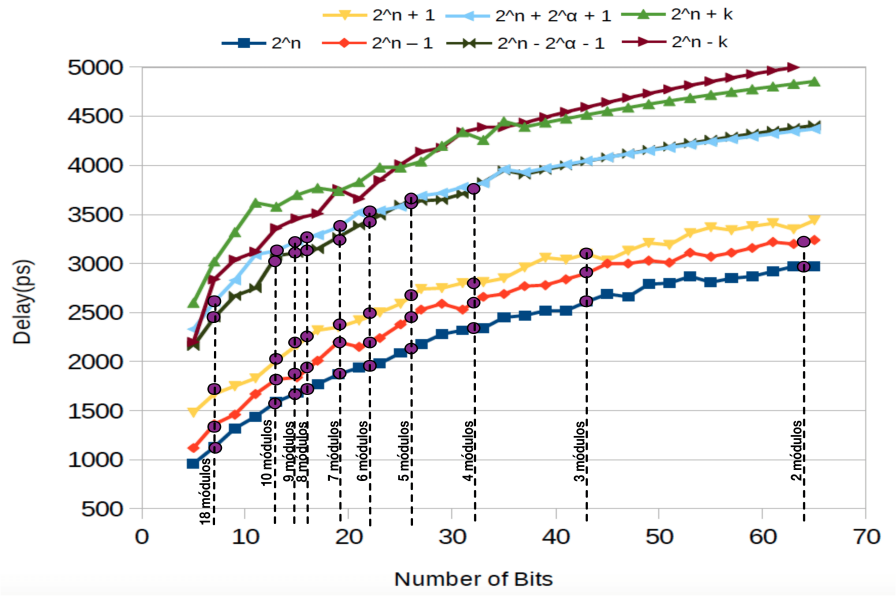
qtd. mod	Δ	Delay(ns)	DR	n
2	$2^n - 1$	3,22	128	64
3	$2^n + 1$	3,10	129	43
4	$2^n - 2^a - 1$	3,78	128	32
5	$2^n + 2^a + 1$	3,63	130	26
6	$2^n + 2^a + 1$	3,53	132	22
7	$2^n + 2^a + 1$	3,37	133	19
8	$2^n + 2^a + 1$	3,25	128	16
9	$2^n + 2^a + 1$	3,23	135	15
10	$2^n + k$	3,58	130	13
11	$2^n + k$	3,62	121	11
12	$2^n + k$	3,62	132	11
13	$2^n + k$	3,50	130	10
14	$2^n + k$	3,32	126	9
15	$2^n + k$	3,32	135	9
16	$2^n + k$	3,13	128	8
17	$2^n + k$	3,13	136	8
18	$2^n + k$	3,02	126	7

4.1.2 Seleção de conjunto de módulos usando $2^n, 2^n \pm 1, 2^n \pm 2^\alpha \pm 1$

A figura 24 mostra que aumentando o número de canais o caminho crítico tende a diminuir. Porém, ao implementar a arquitetura $2^n \pm k$, o caminho crítico adiciona um alto *offset*. É proposto então o mesmo processo, porém neste

caso as unidades $2^n \pm k$ não serão utilizadas para constituir conjuntos de módulos.

Figura 24 – Atraso por número de bits(RNS)



Fonte: produzido pelo autor

Os conjuntos dessa subseção são identicos aos da subseção anterior até $qtd.mod = 9$, mudanças só ocorrem a partir de conjuntos de dez módulos. Onde:
$\{2^n; 2^n - 1; 2^n + 1; 2^n - 3; 2^n + 3; 2^n + 5; 2^n + 15; 2^n + 17; 2^n + 31; 2^n - 33\}$

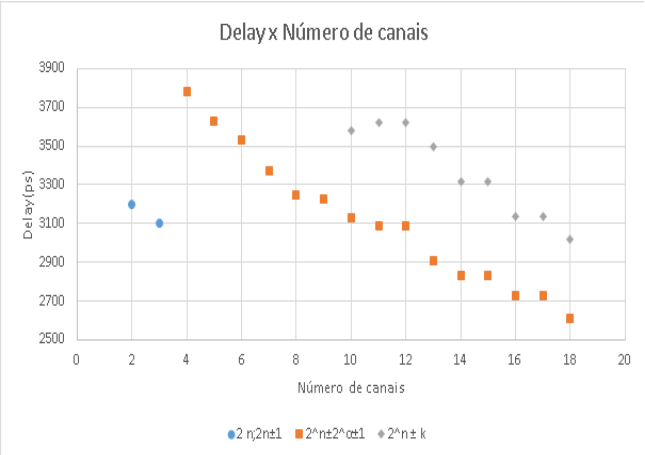
A tabela 6 mostra os novos resultados do caminho crítico, para $10 \leq qtd.mod \leq 18$.

Tabela 6 – Caminho crítico 4.1.2

qtd. mod	Δ	Delay(ns)	DR	n
10	$2^n + 2^\alpha + 1$	3,13	130	13
11	$2^n + 2^\alpha + 1$	3,09	121	11
12	$2^n + 2^\alpha + 1$	3,09	132	11
13	$2^n + 2^\alpha + 1$	2,90	130	10
14	$2^n + 2^\alpha + 1$	2,83	126	9
15	$2^n + 2^\alpha + 1$	2,83	135	9
16	$2^n + 2^\alpha + 1$	2,73	128	8
17	$2^n + 2^\alpha + 1$	2,73	136	8
18	$2^n + 2^\alpha + 1$	2,61	126	7

Na figura 25 encontra-se o mesmo gráfico da subseção anterior, porém adicionando o cenário da exclusão dos módulos $2^n \pm k$. Observa-se uma grande redução no caminho crítico para $10 \leq qtd.mod \leq 18$.

Figura 25 – Caminho crítico por número de canais - 2



Fonte: produzido pelo autor.

Comparando os dois primeiros métodos, o menor caminho crítico oferecido pela primeira opção é de $3,02ns$, enquanto o do segundo método apresentado é de $2,61ns$. Havendo redução de $13,5\%$

4.1.3 Seleção de conjunto de módulos usando 2^n, $2^n \pm 1$

Mesmo havendo uma redução significativa do caminho crítico com a remoção dos módulos $2^n \pm k$, a solução apresentada acima pode apresentar algumas dificuldades de implementação. Pois para obter-se um caminho crítico menor, é necessária uma quantidade muito alta de módulos. Isso pode implicar em:

- Alta complexidade de arquitetura do conversor reverso, que pode implicar em um grande *offset*, assim inviabilizando o uso de um conjunto de módulos com caminho crítico reduzido.

- Dificuldade na escolha de módulos coprimos, visto que os módulos $2^n \pm 2^\alpha \pm 1$ são casos especiais.

A opção proposta então é descartar também o uso de módulos $2^n \pm 2^\alpha \pm 1$. Utilizando somente 2^n e $2^n \pm 1$.

Para que não se use apenas conjuntos de três módulos, a literatura propõe extensões horizontais utilizando $2^n + 1$. Porém um artigo está sendo finalizado pelo grupo de pesquisa da UFSC liderado pelo Prof. Dr. Hector Pettenghi Roldán, baseado em [15], que propõe extensões horizontais utilizando módulos $2^n - 1$. Isso é importante pois esses módulos possuem atrasos reduzidos em relação à $2^n + 1$.

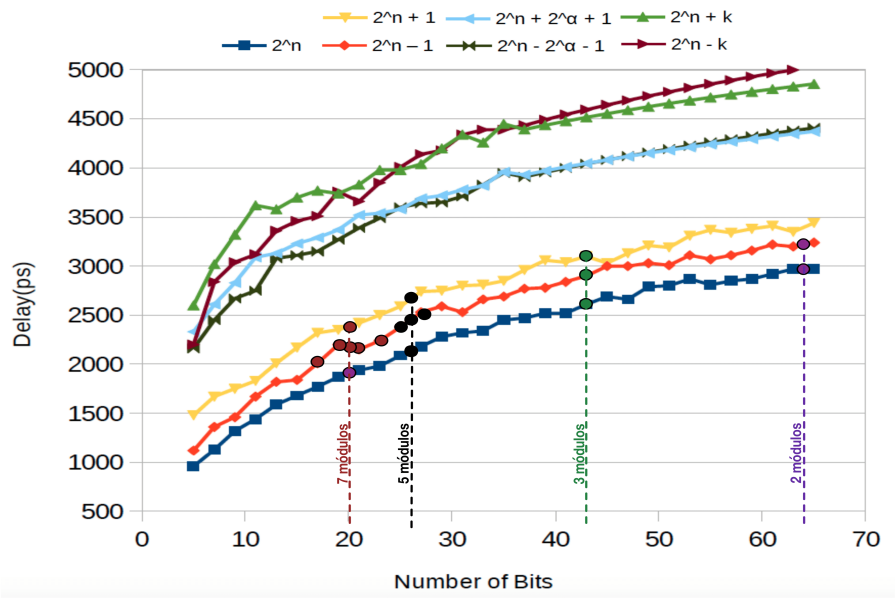
Estão sendo propostos os seguintes conjuntos 7 de módulos:

- $M_1 = \{2^n, 2^n \pm 1, 2^{n\pm1} - 1, 2^{n\pm3} - 1\}$ se $n \neq 6k + 12$
- $M_2 = \{2^n, 2^n \pm 1, 2^{n\pm1} - 1, 2^{n\pm5} - 1\}$ se $n = 6k + 12$

Onde $k \in \mathbb{N}_0$.

Aplicando o método mostrado à $DR = 128bits$, o gráfico abaixo é apresentado:

Figura 26 – Atraso por número de bits(RNS)



Fonte: produzido pelo autor

As tabelas abaixo demonstram exatamente quais são os módulos utilizados:

Tabela 7 – Conjuntos de módulos 4.1.3

qtd. mod	conjunto
2	$2^n; 2^n - 1$
3	$2^n; 2^n - 1; 2^n + 1$
5	$2^n; 2^n - 1; 2^n + 1; 2^{n-1} - 1; 2^{n+1} - 1$
7	$2^n; 2^n - 1; 2^n + 1; 2^{n-1} - 1; 2^{n+1} - 1; 2^{n-3} - 1; 2^{n+3} - 1$

Fonte: produzido pelo autor.

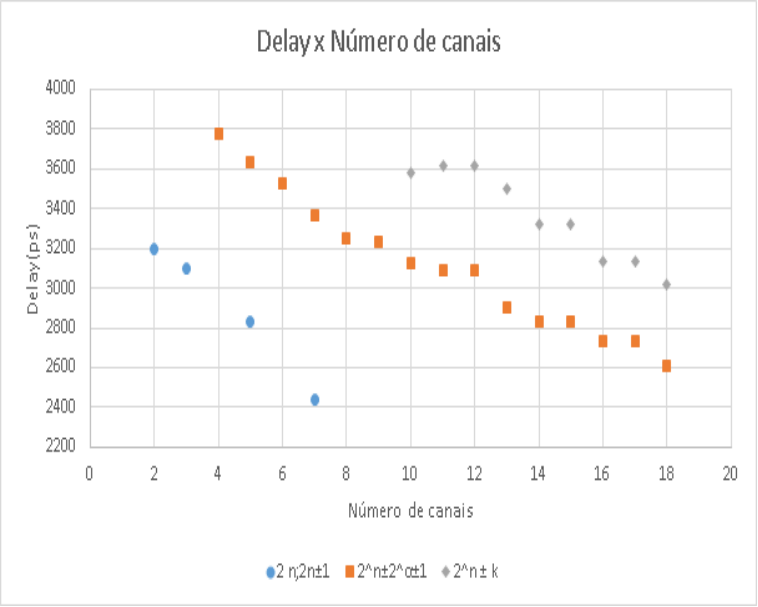
Tabela 8 – Caminho crítico 4.1.4

qtd. mod	Δ	Delay(ns)	DR	n
2	$2^n - 1$	3,22	128	64
3	$2^n + 1$	3,10	129	43
5	$2^n + 1$	2,83	130	26
7	$2^n + 1$	2,44	140	20

Comparando os valores apresentados nesse método com o canal mais otimizado até então $(2,61ns)$, tem-se uma redução de $6,5\%$

Adicionando ao gráfico de caminho crítico por quantidade de módulos:

Figura 27 – Caminho crítico por número de canais - 3



Fonte: produzido pelo autor

A partir do gráfico apresentado, é possível afirmar que esse método é muito mais eficiente em relação aos outros dois apresentados. Pois são utilizados menos canais, os módulos são mais simples e o caminho crítico é reduzido.

4.1.4 Seleção de conjunto de módulos usando $2^n, 2^n - 1$

Na tabela 7 é mostrado que, a partir de três módulos o caminho crítico é o módulo $2^n + 1$. Em uma tentativa

de aceleram ainda mais as operações aritiméticas, o módulo $2^n + 1$ deixará de ser utilizado.

No trabalho que está sendo finalizado, sobre extensões horizontais em $2^n - 1$, são propostos os seguintes conjuntos de oito módulos:

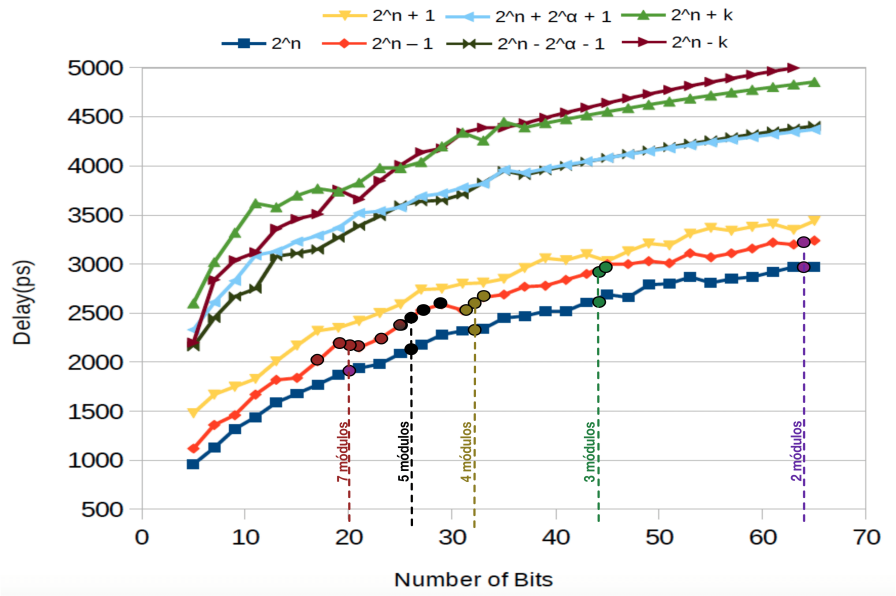
- $M1a = \{M_1, 2^{n+5} - 1\}$ com $n = 6k + 8$ e $k \neq 5\phi + 2$
- $M1b = \{M_1, 2^{n-7} - 1\}$ com $n = 6k + 8$ e $k = 5\phi + 2$
- $M1c = \{M_1, 2^{n-5} - 1\}$ com $n = 6k + 10$ e $k \neq 5\phi$
- $M1d = \{M_1, 2^{n+7} - 1\}$ com $n = 6k + 10$ e $k = 5\phi$

Onde $k \in \mathbb{N}_0$.

Basta então selecionar o conjunto mais adequado e remover o módulo $2^n + 1$.

Aplicando esse método encontra-se o gráfico da página seguinte:

Figura 28 – Atraso por número de bits(RNS)



Fonte: produzido pelo autor

As tabelas 9 e 10 demonstram exatamente quais são os módulos utilizados e os valores de caminho crítico.

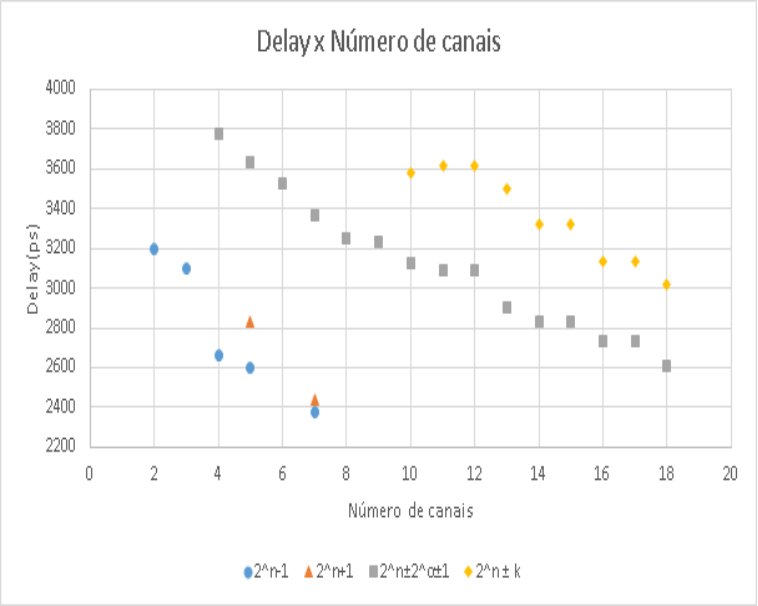
Tabela 9 – Conjuntos de módulos 4.1.4

qtd. mod	conjunto
2	$2^n; 2^n - 1$
3	$2^n; 2^n - 1; 2^{n+1} - 1$
4	$2^n; 2^n - 1; 2^{n+1} - 1; 2^{n-1} - 1;$
5	$2^n; 2^n - 1; 2^{n+1} - 1; 2^{n-1} - 1; 2^{n+3} - 1$
7	$2^n; 2^n - 1; 2^{n-3} - 1; 2^{n-1} - 1; 2^{n+1} - 1; 2^{n+3} - 1; 2^{n+5} - 1$

Tabela 10 – Caminho crítico 4.1.4

qtd. mod	Δ	Delay(ns)	DR	n
2	$2^n - 1$	3,22	128	64
3	$2^n - 1$	3,10	129	43
4	$2^n - 1$	2,66	128	32
5	$2^n - 1$	2,60	130	26
7	$2^n - 1$	2,38	140	20

Figura 29 – Caminho crítico por número de canais - 4



Fonte: produzido pelo autor

Por fim, ao utilizar as técnicas enunciadas, encontra-se que para o caso $DR = 128bits$ e para os métodos utilizados, o conjunto com caminho crítico mais reduzido é $\{2^{20}; 2^{20}-1; 2^{20-3}-1; 2^{20-1}-1; 2^{20+1}-1; 2^{20+3}-1; 2^{20+5}-1\}$, apresentando um atraso de $2,38ns$.

4.2 FORMAS DE OTIMIZAÇÃO PARA AS DIFERENTES FORMAS DE SELEÇÃO APRESENTADAS.

4.2.1 Método iterativo

Em um conjunto de módulos só pode haver um módulo par e que independentemente do valor do seu expoente, esse módulo se manterá coprimo em relação aos demais.

Então, em um conjunto de módulos, onde os módulos ímpares são coprimos para todo n aplicado, pode-se realizar um método iterativo em que se subtrai dos expoentes ímpares um valor i, e a soma desses valores é adicionado ao expoente par.

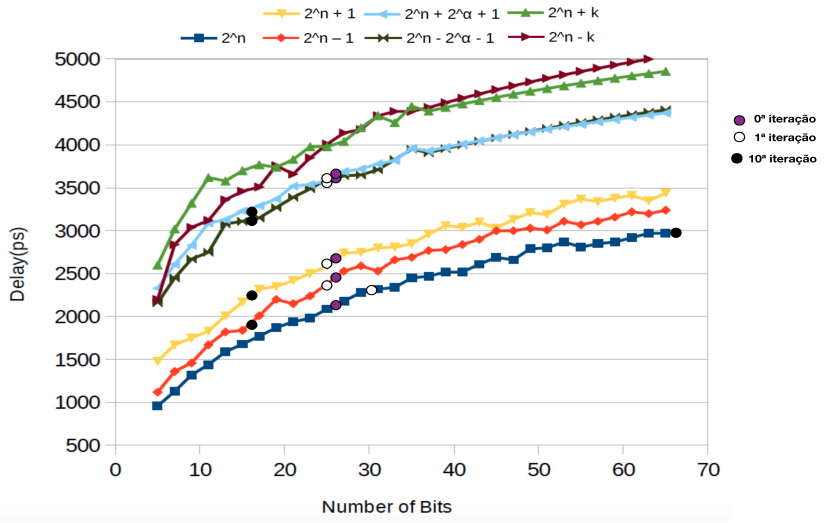
Por exemplo, adotando o caso $DR = 128bits$, e distribuindo a faixa dinâmica entre cinco módulos balanceado. Logo:
$Conjunto_{i=0} = \{2^{26}, 2^{26} \pm 1, 2^{26} \pm 3\}$
$\Delta_{conj._{i=0}} = 3,63ns$
$\Delta_{2^{26}} = 2,14ns$

$$Conjunto_i = \{2^{n+4i}, 2^{n-i} \pm 1, 2^{n-i} \pm 3\} \tag{4.1}$$

$Conjunto_{i=1} = \{2^{30}, 2^{25} \pm 1, 2^{25} \pm 3\}$
$\Delta_{conj._{i=1}} = 3,60ns$
$\Delta_{2^{30}} = 2,30ns$

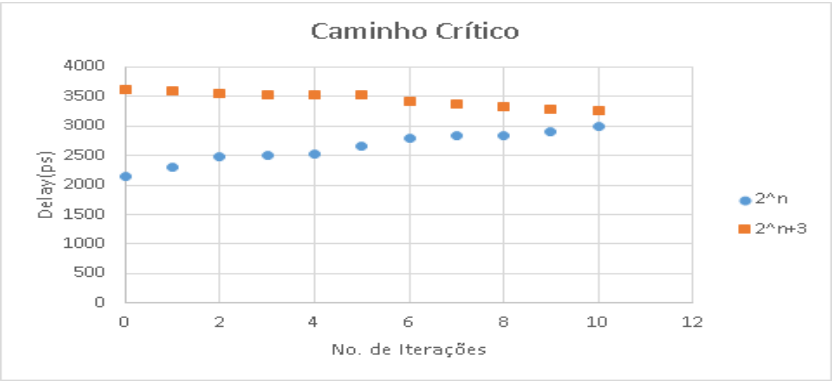
Os gráficos 30 e 31 mostram o comportamento desses módulos após dez iterações: redução de $10,4\%$

Figura 30 – Deslocamento 2^n horizontal - 1



Fonte: produzido pelo autor

Figura 31 – Deslocamento 2^n horizontal - 2



Fonte: produzido pelo autor

É perceptível que preservando a quantidade de módulos, existe uma zona onde o atraso de 2^{n+4i} será o caminho crítico do conjunto.

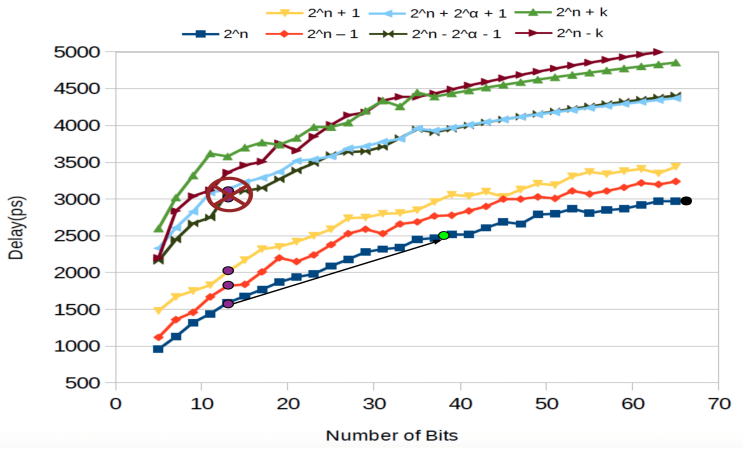
Dessa maneira o caminho crítico é reduzido sem que novos módulos sejam adicionados.

4.2.2 Método direto

Adotando os mesmos formatos de módulos iniciais do exemplo iterativo, porém com $DR = 64bits$. O método direto elimina módulos com atrasos muito altos e redistribui a faixa dinâmica diretamente em 2^n.

Utilizando o exemplo a seguir:
$Q_{inicio} = \{2^{13}, 2^{13} \pm 1, 2^{13} \pm 3\}$, $\Delta = 3,13ns$
$Q_{final} = \{2^{39}, 2^{13} \pm 1\}$, $\Delta = 2,52ns$
$red. = 19,16\%$

Figura 32 – Deslocamento 2^n horizontal direto



Fonte: produzido pelo autor

Ao final desse capítulo foram analisadas diferentes soluções para otimização referente ao caminho crítico. Essas soluções servem para exemplificar maneiras de raciocínio sobre problemáticas RNS em geral. A partir do demonstrado, abrem-se novos caminhos para novos trabalhos, técnicas e idéias com relação a solucionar problemas em RNS.

É importante ressaltar também que algumas técnicas exibidas neste capítulo são novas no estado da arte.

5 CONCLUSÃO

5.1 CONSIDERAÇÕES FINAIS

Ao fim do trabalho, pode se dizer que os objetivos iniciais foram alcançados. E que apesar de não ser o objetivo principal do projeto, obteve-se uma demonstração implementativa/quantitativa da eficácia dos módulos RNS.

Também foi levantada a questão da necessidade de uma análise acerca dos requisitos do projeto, junto uma análise dos módulos disponíveis, para que as unidades RNS trabalhem de maneira otimizada. Ou seja, uma seleção de conjunto de módulos adequados ao projeto para que este funcione de maneira ótima.

No capítulo sobre seleção de módulos, através de análises gráficas/númericas, foram propostas quatro técnicas de seleção de módulos e duas técnicas de otimização. Todas as seis técnicas são focadas na redução do caminho crítico do bloco aritmético RNS, ao mesmo tempo preservando a faixa dinâmica previamente apresentada. Apesar de que em alguns casos o DR tenha sido expandido, esse não se apresentou como foco do trabalho. Caso seja desejado exclusivamente o aumento da faixa dinâmica, deve ser realizada outro tipo de análise

Ao fim do capítulo seis, foi visto que devido ao alto atraso dos módulos $2^n \pm k$ e $2^n \pm 2^\alpha \pm 1$, estes acabaram por ser eliminados. Pois demandavam grande número de canais para a redução de caminho crítico, quando comparados às duas últimas técnicas de seleção de módulos. Podendo então aumentar a complexidade do conversor reverso sugerido, causando um alto *offset* ao caminho crítico.

É esperado também que esse trabalho possa servir de

introdução aos métodos de seleção de módulos para pesquisadores iniciantes na área, exibindo uma linha de raciocício em seleção de módulos baseada em atraso. Apesar de não ter sido enfoque do trabalho, foram apresentados também resultados relacionados à área e potência.

5.2 SUGESTÕES PARA TRABALHOS FUTUROS

- A primeira indicação para trabalhos futuros pode ser a de um estudo sobre os efeitos do conversor reverso sobre o caminho crítico de conjuntos de módulos apresentados(multiplicação por variável). Este seria de grande interesse para que se conheçam o comportamento de *reverse* para as mais variadas combinações de módulos e aplicadas às mais variadas faixas dinâmicas.

- O texto apresentado é um estudo de como tomar decisões para melhorar o caminho crítico do conjunto de módulos de uma maneira geral, mas também existem fatores que são limitantes a outros projetos. Por exemplo: área, potência, *area-delay product(ADP)* e *DR*. Trabalhos realizando análise acerca desses outros parâmetros de desempenho seriam interessantes, como também trabalhos que avaliem esse parâmetros em conjunto.

- Uma outra aplicação interessante é a de tomar decisões sobre conjuntos de módulos de maneira computacional automática. Definindo entradas como: modo de otimização, n^o de bits, faixa dinâmica, Δ_{max}. Com o programa retornando o conjunto de módulos ótimo para a aplicação.

- No texto foi mostrado que os módulos $2^n \pm k$ e $2^n \pm 2^\alpha \pm 1$ sintetizados eram muito piores em velocidade que os

módulos 2^n e $2^n \pm 1$. Sabendo disso, um estudo sobre otimização de módulos com grandes atrasos mudaria a maneira como foi vista a seleção de módulos nesse trabalho.

REFERÊNCIAS

[1] H. Garner; **The residue number system**, IRE Transactions on Electronic Computer; Vol. EC-8; pp.140-147; (1959).

[2] G. Cardarilli, A. Nannarelli, M. Re; **Reducing power dissipation in FIR filters using the residue number system**: IEEE Proceedings of Midwest Symposium on Circuits and Systems; Vol. 1; pp. 320-323, (2000).

[3] S. Piestrak; **Self-testing checkers for arithmetic codes with any check base A**: Proceedings on Pacific Rim International Symposium on Fault-Tolerant Systems; pp. 162-167; (1991).

[4] J. Ramirez, A. garcia, U. Meyer-Baese, A. Lloris; **Fast RNS FPL-Based Communications Receiver Design and Implementation**: Proceedings on 12th International Conference of Field Programmable Logic; pp. 472-481; (2002).

[5] J. Bajard, and L. Imbert; **A Full RNS Implementation of RSA**: IEEE Transactions on Computers; Vol. 53; pp. 769-774; (2004).

[6] H. Nozaki, M. Motoyama, A. Shimbo, and S. Kawamura; **Implementation of RSA algorithm based on RNS montgomery multiplication.** Lecture Notes in Computer Science: Advances in Cryptology - Cryptographic Hardware and Embedded Systems - CHES 2001, pages 364–376. Springer Berlin/Heidelberg, Berlin, Heidelberg, (2001).

[7] S. Antão, J.-C. Bajard, and L. Sousa; **RNS based elliptic curve point multiplication for massive parallel architectures**. The Computer Journal 2011 - Oxford Jour-

nals, pages 1–19, (2011).

[8] H. Pettenghi, R. Chaves and L. Sousa; **Method for designing Multi-Channel RNS Architectures to prevent Power Analysis SCA**. Int. Symp. on Circuits and Syst. (ISCAS14), Melbourne, (2014).

[9] R. Chaves and L. Sousa;**$2^n + 1, 2^{n+k}, 2^n - 1$: A New RNS Moduli Set Extention**, In IEEE Euromicro Symposium on Digital System Design: Architectures, Methods and Tools, IEEE Computer Society, pp. 210-217, Rennes, France, September 2004.

[10] R. Zimmermann; **Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication**: Proceedings of the 14th IEEE Symposium on Computer Architecture; pp. 158-167; (1999).

[11] P. M. Matutino, R. Chaves, and L. Sousa; **Arithmetic Units for RNS moduli $2^n - 3$ and $2^n + 3$ operations**. 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools; pp. 243-246; (2010).

[12] P. M. Matutino, H. Pettenghi, R. Chaves and L. Sousa; **RNS Arithmetic units for modulo $2^n \pm k$**, In 15th Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD2012), Izmir (Turkey), (2012).

[13] H. Pettenghi, S. Cotofana and L. Sousa; **Efficient Method for Designing modulo $2^n \pm k$ multipliers**. Journal of Circuits, Systems, and Computers, v. 23, pp. 1450001(1)-1450001(20), (2013).

[14] N. Keivan, A. S. Molahosseini and M. Esmaeildoust; **How to teach residue number system to computer scientists and engineers**. IEEE Transactions on Education; Vol. 54.1 : pp. 156-163; (2011).

[15] B. Cao, C.-H. Chang, and T. Srikanthan; **A residue-to-binary converter for a new five-moduli set.** IEEE Transactions on Circuits and Systems I: Regular Papers, 54(5):1041–1049, (2007).

APÊNDICE A – Resultados sínteses, equações de regressão e análise temporal caso 1

A.1 SÍNTESE BINÁRIO

n	timing(ns)	area(m2)	power(mW)
1	190	6,84	0,23452
2	250	32,76	0,49135
3	410	104,04	0,80578
4	560	328,31	1,2
5	740	326,88	1,62
6	860	553,32	2,25
7	920	715,68	2,81
8	1150	978,84	3,6
9	1370	1120,32	4,33
10	1490	1483,2	5,44
11	1760	1517,04	5,94
12	1840	1997,64	7,37
13	1930	2526,12	9
14	2250	2448,36	9,04
15	2360	2873,52	10,58
16	2450	3378,96	11,91
17	2640	3895,56	13,79
18	2770	4068,72	14,63
19	2890	4712,4	16,51
20	3100	5011,2	18,12
21	3310	5187,24	18,44
22	3420	6185,52	21,86
23	3610	6378,48	22,53
24	3810	7041,6	24,51
25	4050	7372,8	25,98
26	4310	7905,96	27,65
27	4380	8960,04	31,13
28	4590	9173,16	32,6
29	4800	9728,64	33,85
30	5090	9831,96	34,67
31	5050	11394,72	39,97
32	5210	12177,72	42,38
33	5650	11995,2	42,83
34	5600	13584,6	47,5
35	5790	13802,04	47,92
36	5940	15129	53,63
37	6120	15436,08	54,28
38	6190	16596,72	57,61
39	6440	17246,52	59,97
40	6540	18403,2	64,19
50	8530	26596,8	93,38
75	13220	57193,56	199,53
100	18070	97156,8	339,71
150	27050	218500,92	764,13
200	36610	382871,16	1342

A.2 SÍNTESE RNS *DELAY* MÓDULOS $\mathbf{2^N}$, $\mathbf{2^N-1}$, $\mathbf{2^N+1}$

	Delay(ps)		
n	$\mathbf{2^n}$	$\mathbf{2^n-1}$	$\mathbf{2^n+1}$
5	960	1120	1480
7	1130	1360	1670
9	1320	1460	1750
11	1440	1670	1830
13	1590	1820	2010
15	1680	1840	2170
17	1770	2010	2320
19	1870	2200	2350
21	1940	2150	2420
23	1980	2240	2500
25	2090	2380	2590
27	2180	2530	2740
29	2280	2590	2750
31	2320	2530	2800
33	2340	2660	2810
35	2450	2690	2850
37	2470	2770	2960
39	2520	2780	3060
41	2520	2840	3040
43	2610	2900	3100
45	2690	3000	3030
47	2660	3000	3130
49	2790	3030	3210
51	2800	3010	3190
53	2870	3110	3310
55	2810	3070	3370
57	2850	3110	3340
59	2870	3160	3380
61	2920	3220	3410
63	2970	3200	3350
65	2970	3240	3440

A.3 SÍNTESE RNS *DELAY* MÓDULOS $\mathbf{2^N \pm 2^\alpha \pm 1}$ & $\mathbf{2^N \pm K}$

	Delay(ps)			
n	$\mathbf{2^n + 2^\alpha + 1}$	$\mathbf{2^n - 2^\alpha - 1}$	$\mathbf{2^n + k}$	$\mathbf{2^n - k}$
5	2330	2160	2600	2200
7	2610	2450	3020	2840
9	2830	2670	3320	3040
11	3090	2750	3620	3120
13	3130	3080	3580	3360
15	3230	3109,33	3700	3460
17	3290	3150	3770	3510
19	3370	3270	3740	3760
21	3520	3390	3830	3660
23	3540	3490	3980	3850
25	3580	3600	3980	4010
27	3690	3640	4040	4140
29	3720	3650	4200	4180
31	3780	3710	4340	4340
33	3820	3830	4260	4390
35	3960	3950	4450	4390
37	3930,34	3909,27	4393,17	4435,57
39	3971,56	3955,91	4436,44	4491,37
41	4010,72	4000,22	4477,55	4544,38
43	4048,01	4042,42	4516,71	4594,87
45	4083,61	4082,71	4554,07	4643,06
47	4117,66	4121,23	4589,82	4689,15
49	4150,29	4158,15	4624,07	4733,32
51	4181,61	4193,59	4656,96	4775,73
53	4211,73	4227,67	4688,57	4816,5
55	4240,74	4260,49	4719,02	4855,77
57	4268,7	4292,14	4748,38	4893,63
59	4295,71	4322,69	4776,73	4930,18
61	4321,81	4352,23	4804,13	4965,52
63	4347,07	4380,81	4830,65	4999,72
65	4371,54	4408,50	4856,34	5032,85

A.4 SÍNTESE RNS *POWER* MÓDULOS 2^N, 2^N-1, 2^N+1

	Power(mW)		
n	2^n	2^n-1	2^n+1
5	1,5893	2,5147	3,8832
7	2,8881	4,592	6,7266
9	3,8337	6,8041	10,0811
11	6,2491	9,08	14,186
13	7,6202	12,3607	15,2819
15	9,8081	16,6958	19,657
17	11,762	18,249	22,0963
19	14,8479	22,2604	28,186
21	17,1372	26,5514	32,7891
23	21,0274	32,263	37,8586
25	22,2529	37,0182	46,5473
27	25,677	42,5001	48,78
29	29,2404	47,6979	54,937
31	33,3067	56,2897	62,0878
33	37,9704	61,2764	68,9579
35	41,311	67,9181	73,1061
37	45,3813	70,8137	85,6895
39	47,7461	80,4533	90,4801
41	54,2847	89,5325	100,8934
43	59,2509	98,6837	106,9533
45	64,7016	105,263	117,5092
47	70,027	110,574	125,4062
49	75,3085	124,2267	136,2049
51	81,9784	134,3388	148,1158
53	84,7335	140,7508	156,8192
55	93,8008	149,4704	161,8517
57	97,7895	161,8309	180,9836
59	102,6688	174,8205	187,0768
61	110,9608	183,0404	203,4677
63	114,0554	195,3946	218,2504
65	125,3962	211,127	228,5126

A.5 RNS SÍNTESES *POWER* $\mathbf{2^N \pm 2^\alpha \pm 1}$ & $\mathbf{2^N \pm K}$

	Power(mW)			
n	$\mathbf{2^n + 2^\alpha + 1}$	$\mathbf{2^n - 2^\alpha - 1}$	$\mathbf{2^n + k}$	$\mathbf{2^n - k}$
5	8,71	5,35	9,64	6,75
7	15,18	11,52	16,41	10,82
9	22,31	16,15	21,61	16,73
11	31,89	24,32	28,07	23,27
13	39,35	29,77	34,77	27,72
15	48,41	36,98	41,02	38,72
17	58,43	45,67	50,57	44,18
19	63,64	54,45	57,62	47,74
21	73,87	63,97	67,15	57,98
23	85,03	74,76	75,94	66,04
25	95,27	85,4	84,41	74,77
27	109,85	96,56	98,35	82,13
29	119,46	107,75	108,36	94,32
31	135,15	119,79	120,5	106,88
33	151,97	135,18	131,42	113,07
35	163,71	153,33	141,82	124,58
37	178,3147	165,4095	158,0202	136,6310
39	193,8783	181,2255	171,4446	148,4566
41	210,0363	197,7335	185,3746	160,7166
43	226,7887	214,9335	199,8102	173,4110
45	244,1355	232,8255	214,7514	186,5398
47	262,0767	251,4095	230,1982	200,1030
49	280,6123	270,6855	246,1506	214,1006
51	299,7423	290,6535	262,6086	228,5326
53	319,4667	311,3135	279,5722	243,3990
55	339,7855	332,6655	297,0414	258,6998
57	360,6987	354,7095	315,0162	274,4350
59	382,2063	377,4455	333,4966	290,6046
61	404,3083	400,8735	352,4826	307,2086
63	427,0047	424,9935	371,9742	324,2470
65	450,2955	449,8055	391,9714	341,7198

A.6 SÍNTESE RNS *AREA* MÓDULOS 2^N, $2^N - 1$, $2^N + 1$

Area (μm^2)			n
2^n	$2^n - 1$	$2^n + 1$	
5	407,52	575,28	932,76
7	929,88	1068,84	1648,44
9	1218,6	1625,4	2431,08
11	2230,56	2193,84	3515,76
13	2697,12	3089,52	3927,24
15	3483,72	4233,24	5050,8
17	4302	4595,4	6046,2
19	5536,08	5694,84	7509,96
21	6327,36	7026,48	8977,32
23	7907,4	8478,36	10692,72
25	8542,44	10368	13287,96
27	10078,2	12152,88	14550,48
29	11596,32	13737,6	16606,8
31	13210,2	16170,12	18665,64
33	15260,76	17908,2	20746,08
35	16786,8	20025,72	22529,16
37	18692,27	21184,91	26183,52
39	20269,79	24204,95	28065,96
41	22652,63	27026,63	31437,72
43	25007,39	29711,16	32864,04
45	27439,19	31962,95	36543,6
47	29918,51	33773,39	39600
49	32437,07	37671,47	43456,68
51	34909,19	40738,32	47029,68
53	37074,59	43692,47	50359,32
55	40888,79	46139,03	53237,88
57	43594,91	50285,51	58652,64
59	45905,39	54042,83	61007,76
61	49915,43	56608,19	65842,2
63	51601,67	60648,11	70725,96
65	56228,75	65926,43	74332,08

A.7 SÍNTESES RNS *AREA* MÓDULOS $2^N \pm 2^\alpha \pm 1$ & $2^N \pm K$

	Area (μm^2)			
n	$2^n + 2^\alpha + 1$	$2^n - 2^\alpha - 1$	$2^n + k$	$2^n - k$
5	3403,8	1960,9	3525,4	3233,8
7	5728,3	4678,2	5751,3	4471,2
9	8019	6111	7548,48	6516
11	12209,04	9288,72	9596,88	8477,28
13	14649,12	11491,2	11960,64	9950,4
15	18873,35	14068,44	14221,08	13355,21
17	22429,44	17383,68	17350,2	15561
19	23047,91	20850,47	19760,39	17229,96
21	27069,11	24567,47	22769,28	20721,6
23	30915,71	28637,27	26192,52	23717,16
25	35426,87	32822,27	29426,75	26732,88
27	41294,15	38494,07	33932,87	30151,08
29	45798,11	43486,19	38445,11	34498,07
31	51588,71	48372,47	42771,95	38779,92
33	57831,83	54291,95	46852,19	42431,03
35	63338,39	61369,19	51058,07	46664,27
37	68892,16	67114,57	56385,20	51420,43
39	75210,39	73851,88	61543,28	56225,14
41	81796,54	80908,20	66922,56	61241,07
43	88650,61	88283,54	72523,04	66468,22
45	95772,61	95977,89	78344,72	71906,60
47	103162,54	103991,25	84387,60	77556,20
49	110820,39	112323,62	90651,68	83417,03
51	118746,16	120975,00	97136,96	89489,08
53	126939,86	129945,39	103843,43	95772,36
55	135401,49	139234,80	110771,11	102266,86
57	144131,03	148843,22	117919,98	108972,58
59	153128,51	158770,65	125290,06	115889,52
61	162393,91	169017,09	132881,33	123017,69
63	171927,23	179582,55	140693,81	130357,09
65	181728,48	190467,01	148727,48	137907,71

A.8 REGRESSÕES BINÁRIO

Tabela 11 – Regressões Binário

Propriedade	Equação
$\boldsymbol{Delay}$	$\mathbf{183,3383}n - \mathbf{454,3784}$
$\boldsymbol{Area}$	$\mathbf{0,03235}n^2 + \mathbf{0,2338}n + \mathbf{0,2478}$
$\boldsymbol{Power}$	$\mathbf{9,2046}n^2 + \mathbf{73,2386}n - \mathbf{92,3435}$

A.9 REGRESSÕES RNS

Tabela 12 – Regressões Delay - RNS

Canal	D(n)
$\mathbf{2}^n$	$\mathbf{836}\ln(n) - \mathbf{540}$
$\mathbf{2}^n - \mathbf{1}$	$\mathbf{875}\ln(n) - \mathbf{413}$
$\mathbf{2}^n + \mathbf{1}$	$\mathbf{824}\ln(n) - \mathbf{25}$
$\mathbf{2}^n + \boldsymbol{k}$	$\mathbf{822}\ln(n) + \mathbf{1425}$
$\mathbf{2}^n - \boldsymbol{k}$	$\mathbf{1060}\ln(n) + \mathbf{608}$
$\mathbf{2}^n + \mathbf{2}^\alpha + \mathbf{1}$	$\mathbf{783}\ln(n) + \mathbf{1103}$
$\mathbf{2}^n - \mathbf{2}^\alpha - \mathbf{1}$	$\mathbf{886}\ln(n) + \mathbf{710}$

Tabela 13 – Regressões Power - RNS

Canal	P(n)
$\mathbf{2^n}$	$\mathbf{0,0238n^2 + 0,3828n - 1,2478}$
$\mathbf{2^n - 1}$	$\mathbf{0,0429n^2 + 0,4169n - 0,3702}$
$\mathbf{2^n + 1}$	$\mathbf{0,0452n^2 + 0,5455n + 1,2483}$
$\mathbf{2^n + k}$	$\mathbf{0,0743n^2 + 2,1350n - 2,3970}$
$\mathbf{2^n - k}$	$\mathbf{0,0865n^2 + 1,3340n - 2,3670}$
$\mathbf{2^n + 2^\alpha + 1}$	$\mathbf{0,0632n^2 + 1.9090n + 0.8660}$
$\mathbf{2^n - 2^\alpha - 1}$	$\mathbf{0,0532n^2 + 1,7860n + 3,7800}$

Tabela 14 – Regressões Area - RNS

Canal	A(n)
$\mathbf{2^n}$	$\mathbf{12,6800n^2 + 35,2600n + 47,3940}$
$\mathbf{2^n - 1}$	$\mathbf{14,5040n^2 + 58,3940n - 176,6000}$
$\mathbf{2^n + 1}$	$\mathbf{16,3440n^2 + 76,5500n + 342,5900}$
$\mathbf{2^n + k}$	$\mathbf{33,4906n^2 + 613,8273n + 331,9239}$
$\mathbf{2^n - k}$	$\mathbf{39,8765n^2 + 338,0414n + 16,1158}$
$\mathbf{2^n + 2^\alpha + 1}$	$\mathbf{27,6499n^2 + 477,6488n + 859,4828}$
$\mathbf{2^n - 2^\alpha - 1}$	$\mathbf{26,4030n^2 + 395,7253n + 632,8935}$

A.10 ANÁLISE TEMPORAL

Tabela 15 – Análise temporal módulo multiplicador caso 1

		Registrador 1			Registrador 2	
Instante	X	Y	X_reg	Y_reg	S_reg	S
t=$\mathbf{0}^-$	A	1	1	1	1	1
t=$\mathbf{0}^+$	A	1	A	1	1	1
t=$\boldsymbol{T}^-$	A	1	A	1	A	1
t=$\boldsymbol{T}^+$	A	1	A	1	A	A
t=$\mathbf{2}\boldsymbol{T}^+$	B	A	A	1	A	A
t=$\mathbf{2}\boldsymbol{T}^+$	B	A	B	A	A	A
t=$\mathbf{3}\boldsymbol{T}^-$	B	A	B	A	A*B	A
t=$\mathbf{3}\boldsymbol{T}^+$	B	A	B	A	A*B	A*B
t=$\mathbf{4}\boldsymbol{T}^-$	C	A*B	B	A	A*B	A*B
t=$\mathbf{4}\boldsymbol{T}^+$	C	A*B	C	A*B	A*B	A*B
t=$\mathbf{5}\boldsymbol{T}^-$	C	A*B	C	A*B	A*B*C	A*B
t=$\mathbf{5}\boldsymbol{T}^+$	C	A*B	C	A*B	A*B*C	A*B*C
t=$\mathbf{6}\boldsymbol{T}^-$	D	A*B*C	C	A*B	A*B*C	A*B*C
t=$\mathbf{6}\boldsymbol{T}^+$	D	A*B*C	D	A*B*C	A*B*C	A*B*C
t=$\mathbf{7}\boldsymbol{T}^-$	D	A*B*C	D	A*B*C	A*B*C*D	A*B*C
t=$\mathbf{7}\boldsymbol{T}^+$	D	A*B*C	D	A*B*C	A*B*C*D	A*B*C*D

Nota-se que o intervalo entre os instantes $\mathbf{2}\boldsymbol{T}^+$ e $\mathbf{3}\boldsymbol{T}^-$, por exemplo, é aquele em que a operação aritmética está a sendo realizada. Neste caso pontual a operação é a de $\boldsymbol{A}*\boldsymbol{B}$.